The MathWorks

MATLAB® & SIMULINK®

Production Code Generation (PCG) Time Machine

Tom Erkkinen

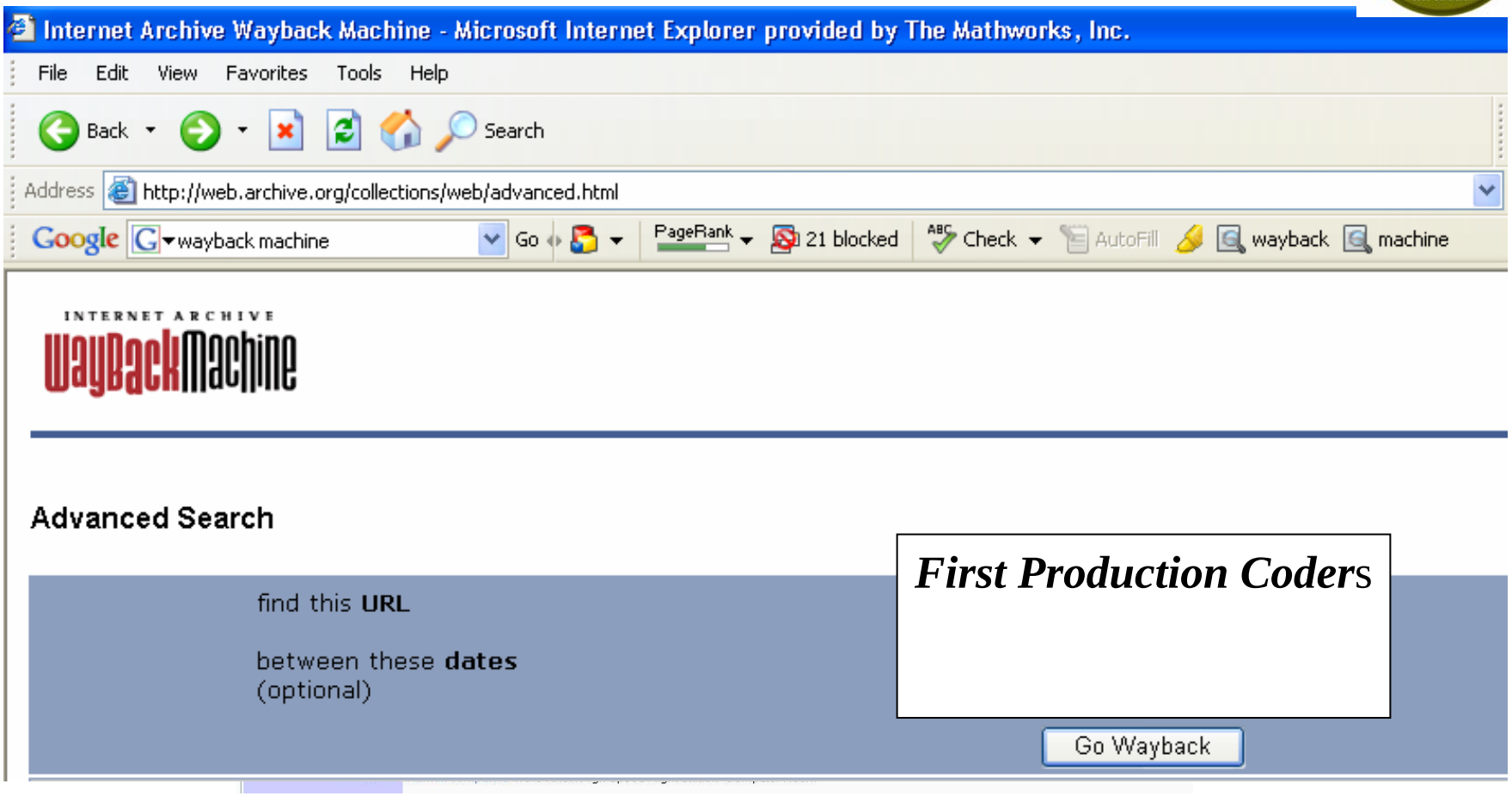
Embedded Applications Manager

The MathWorks, Inc.

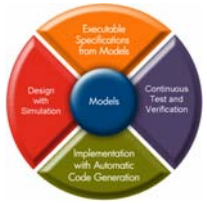
© 2008 The MathWorks, Inc.

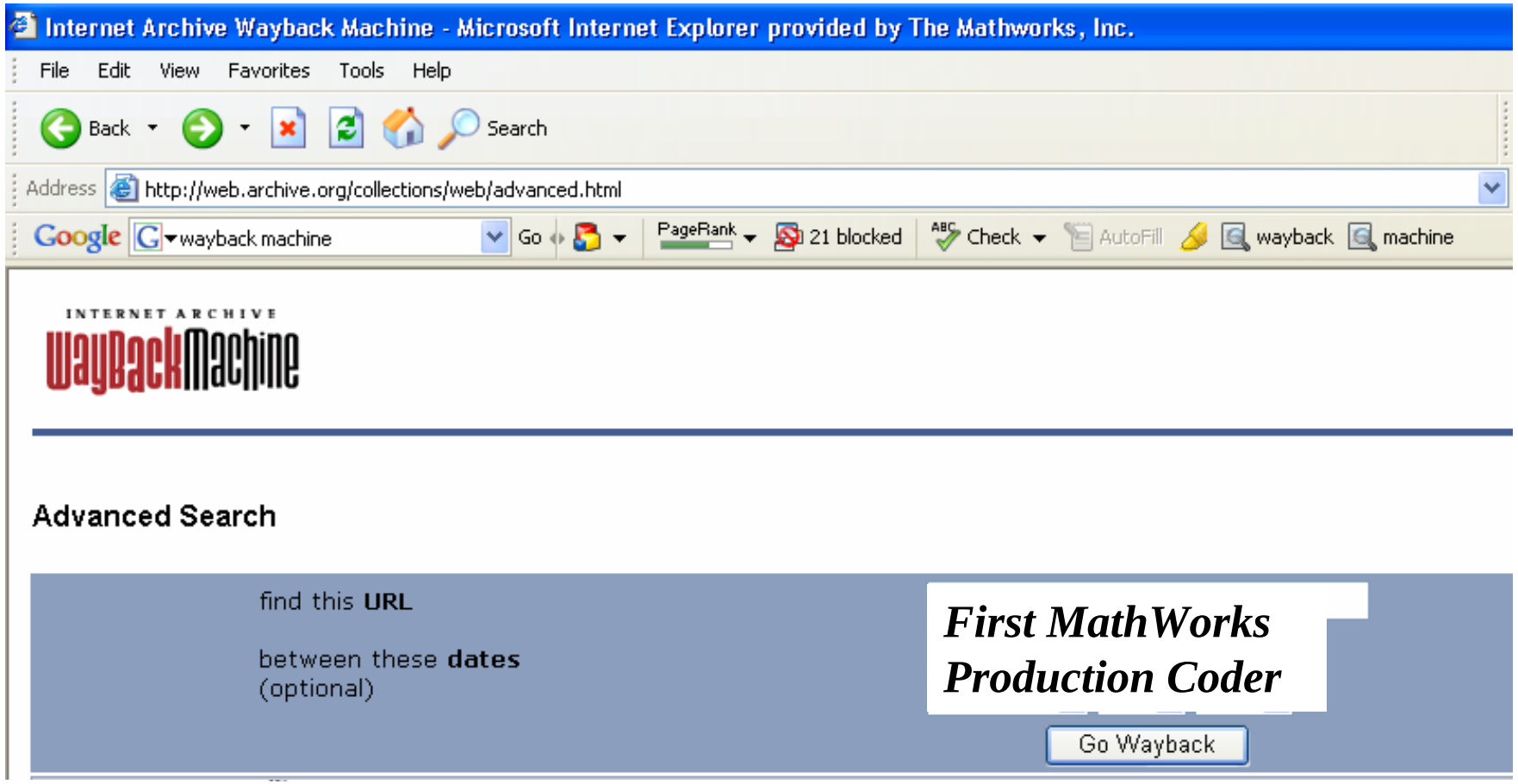
MathWorks Automotive Conference '08

NACA "Production Coders" - 1949

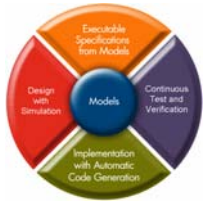


Wayback Machine

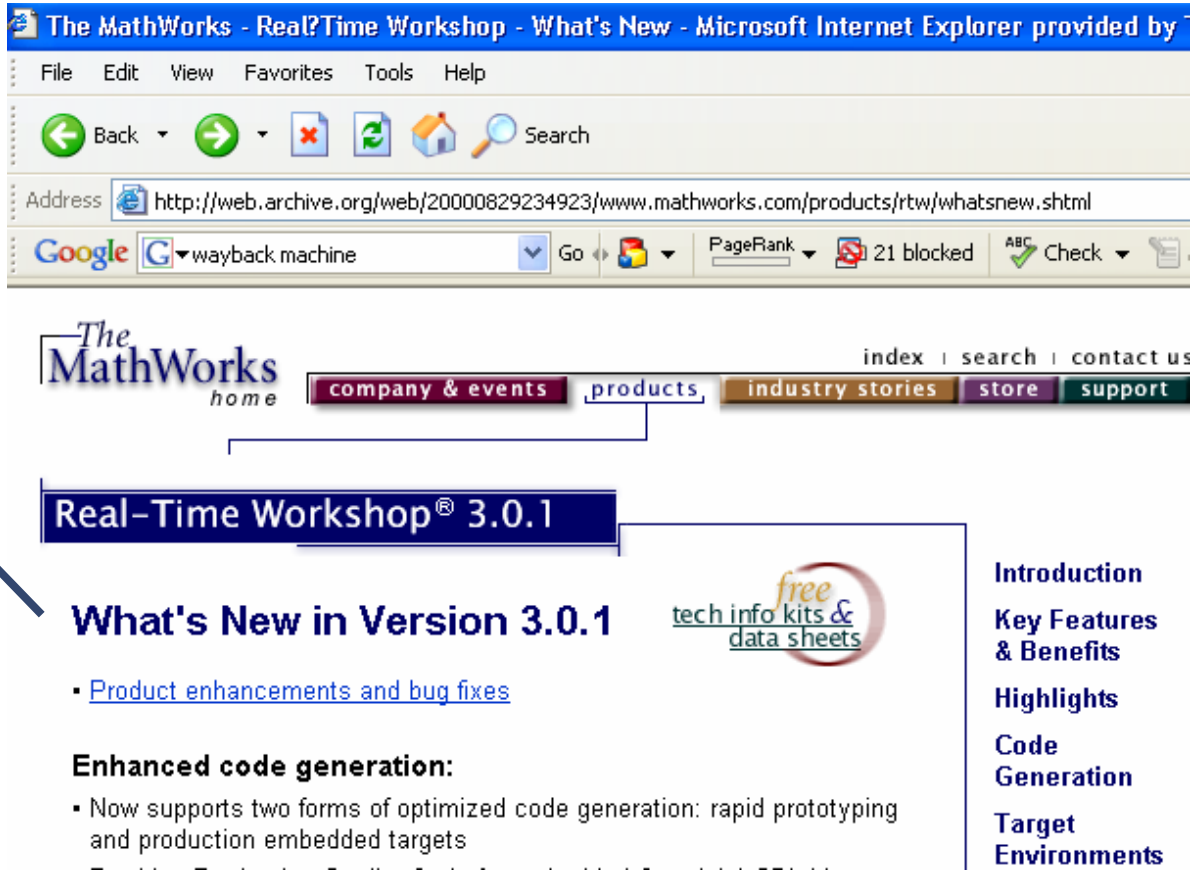




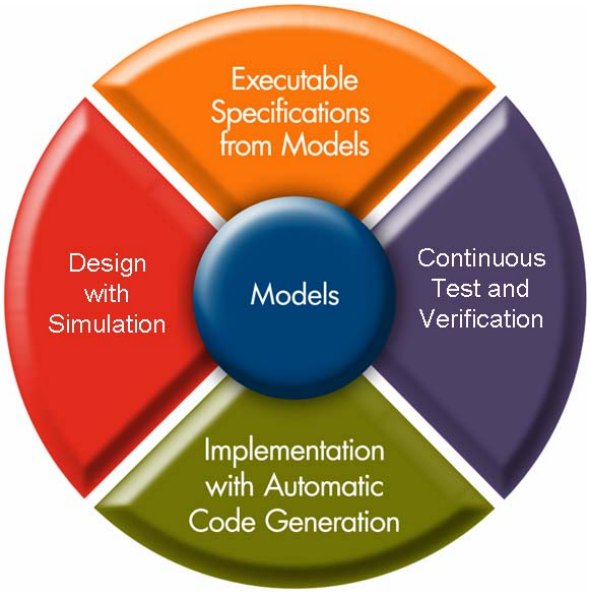
Simulink Code Generation - 1999



"Now supports two forms of code generation:


The MathWorks - Real?Time Workshop - What's New - Microsoft Internet Explorer provided by
File Edit View Favorites Tools Help
Back Search
Address http://web.archive.org/web/20000829234923/www.mathworks.com/products/rtw/whatsnew.shtml
Google wayback machine Go PageRank 21 blocked Check
The MathWorks home
index | search | contact us
company & events
products
industry stories
store
support
Real-Time Workshop® 3.0.1
What's New in Version 3.0.1
free tech info kits & data sheets
Product enhancements and bug fixes
Enhanced code generation:
Now supports two forms of optimized code generation: rapid prototyping and production embedded targets
Introduction
Key Features & Benefits
Highlights
Code Generation
Target Environments

Agenda



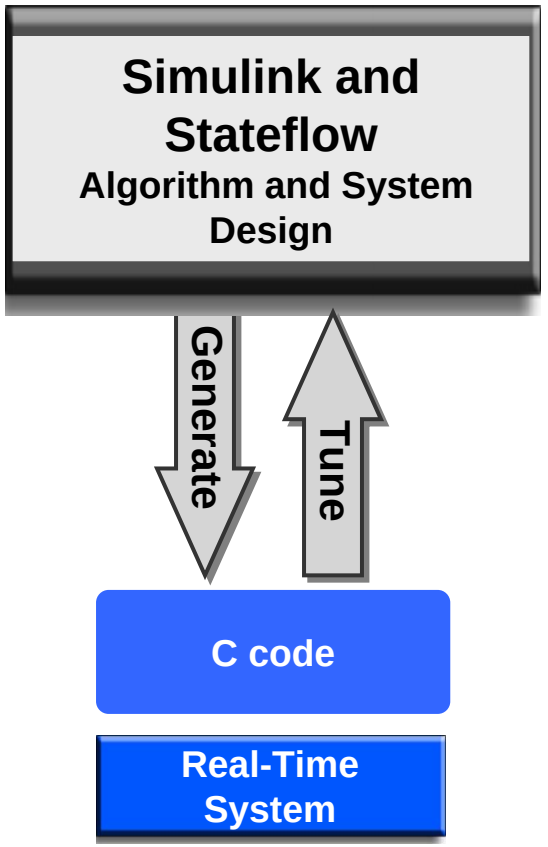
PCG Historical Review

- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

Code Generation – 1990s

Real-Time Workshop® and Stateflow® Coder

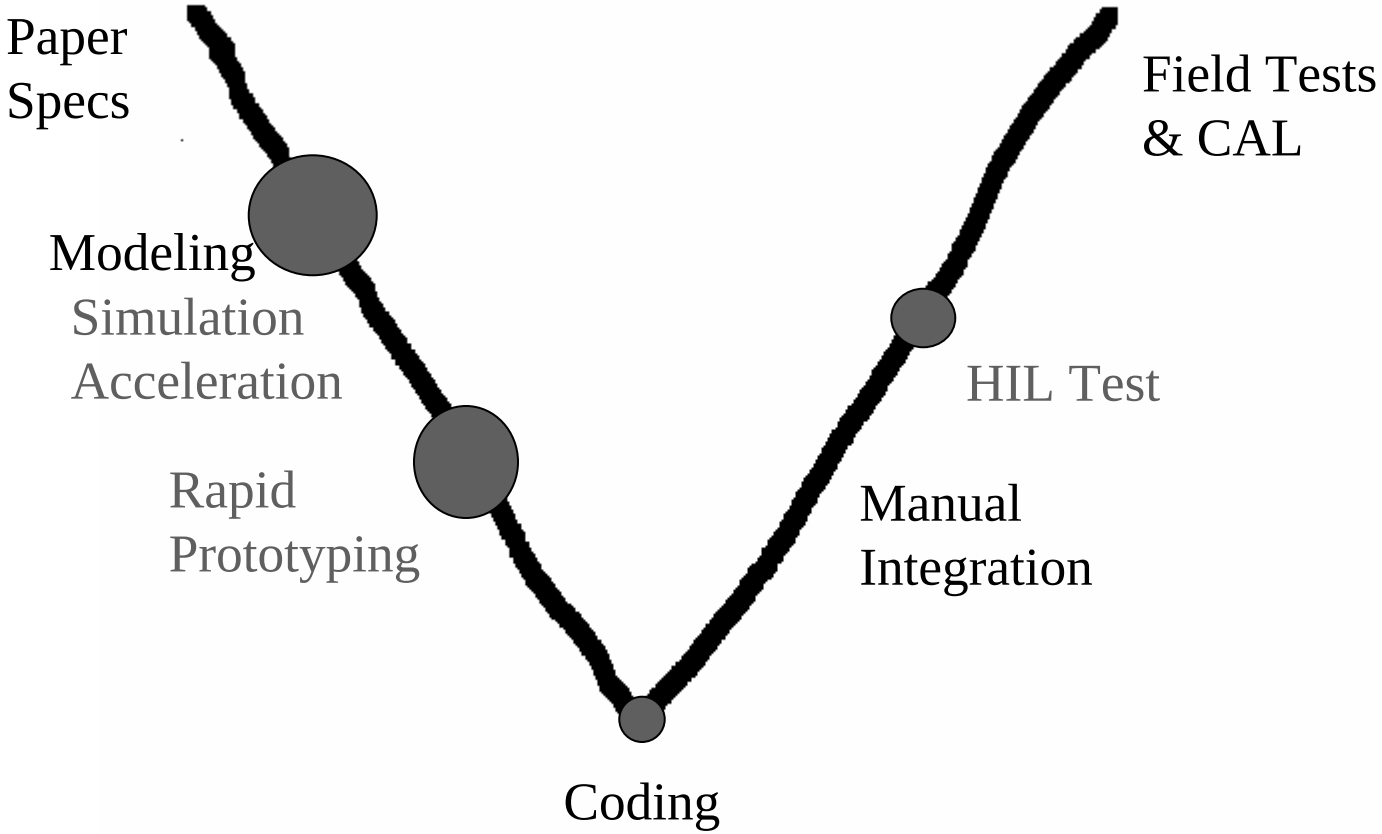
- Generate code from Simulink and Stateflow that is easy to interact, tune, and experiment with
  - Simulation Acceleration
  - Rapid Prototyping
  - Hardware in Loop (HIL)
  - Embedded deployment (some examples)



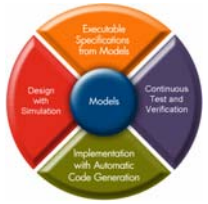
You can deploy code on any microprocessor using Real-Time Workshop because it generates ANSI-C.

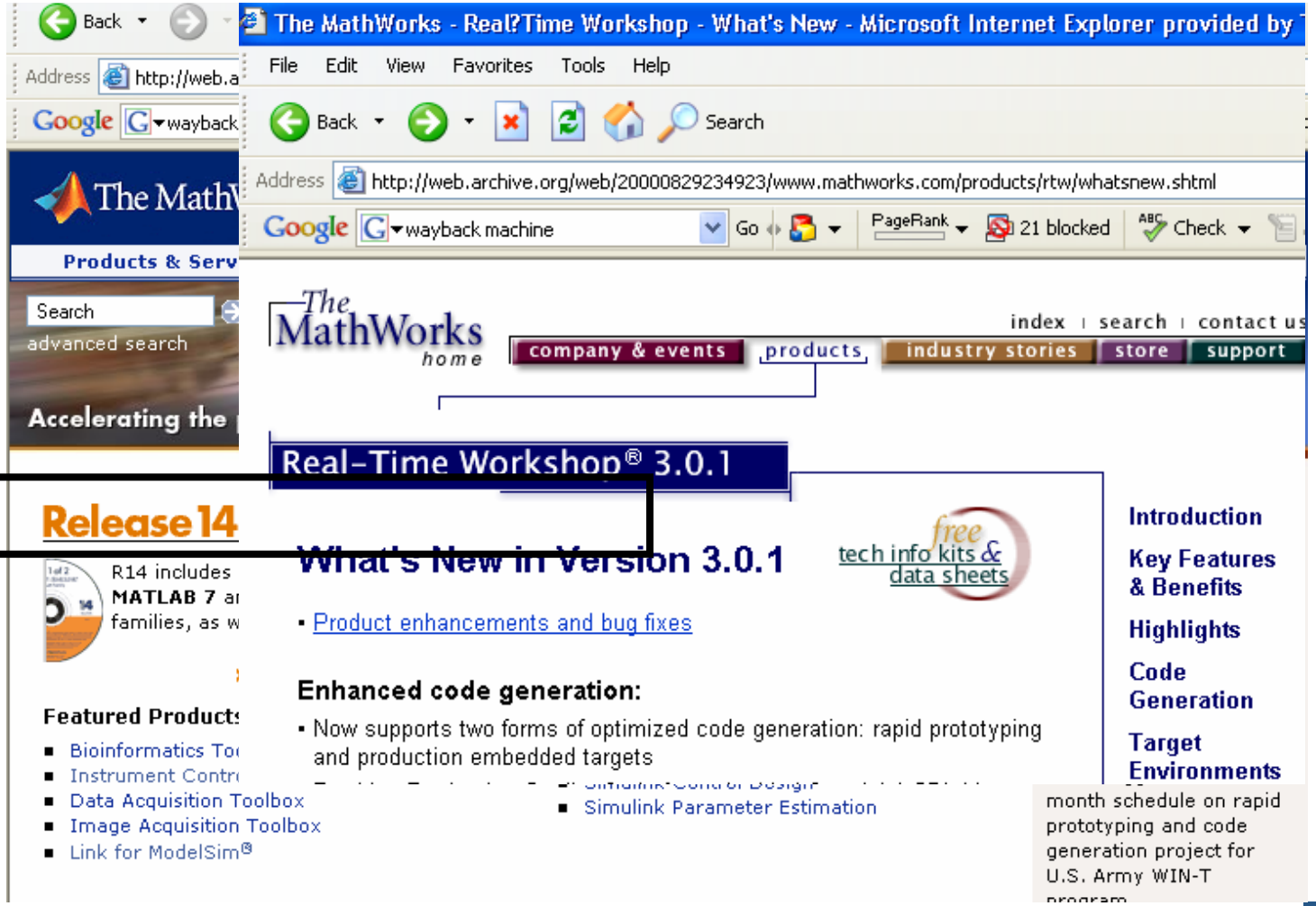
ECU Development – 1990s

Automatic Code Generation usage

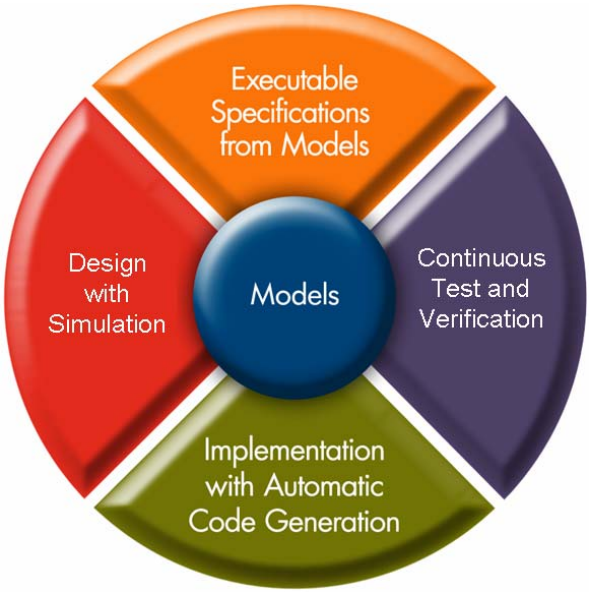


Code Generation (R11)





Agenda



PCG Historical Review

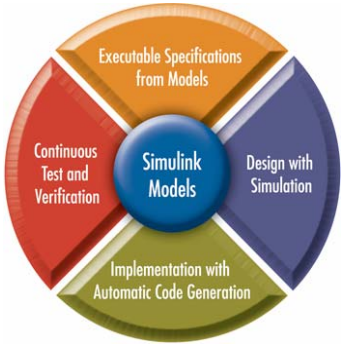
- 1990s (Circa Release 11)
- **2000s (Circa Release 14)**
- Today (R2008a)

New features and trends

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification

ECU Development – 2000s

Using Model-Based Design with Automatic Code Generation



System Specification

Simulation Acceleration

Component Design

Rapid Prototyping

Software Design

On Target Rapid Prototyping

Coding
• Source

Production Code Generation

Software Integration

Software-in-loop (SIL) Test

Component Integration

Processor-in-loop (PIL) Test

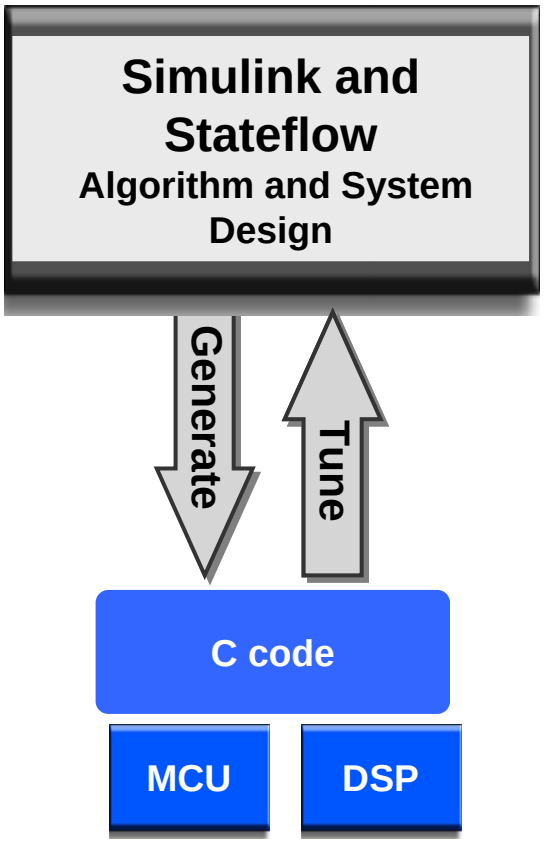
System Integration

HIL Test

Code Generation – Early 2000s

Real-Time Workshop® Embedded Coder

- Supports all Real-Time Workshop options
- Plus generates production code
  - Floating- and fixed-point efficiency
  - Code readability
  - Component integration
  - Golden model



MAC 2005 - GMPT

Pilot Result - Metrics

	Hand Code	Auto-code	Percent Change
Calibration ROM	9464	9464	0%
Code ROM	2952	2900	-1.76%
RAM	240	238	-0.83%

More Efficient Than Hand

MAC 2005 - Visteon

Model-Based Development Progress: Overview

Status:

- Visteon Powertrain is using model-based software development globally
- Production intent programs & Advanced projects
- Model-based process: 5 years
- Production intent auto coding: 3 years

Objective:

- Incrementally deploy MBD rather than "big-bang"
  - Model based components must mix with legacy software
- Improve quality while reducing development time and cost
  - Models & simulation improve quality: better understanding leads to better design
  - Reduce development time by employing auto-code generation, test case generation, etc.

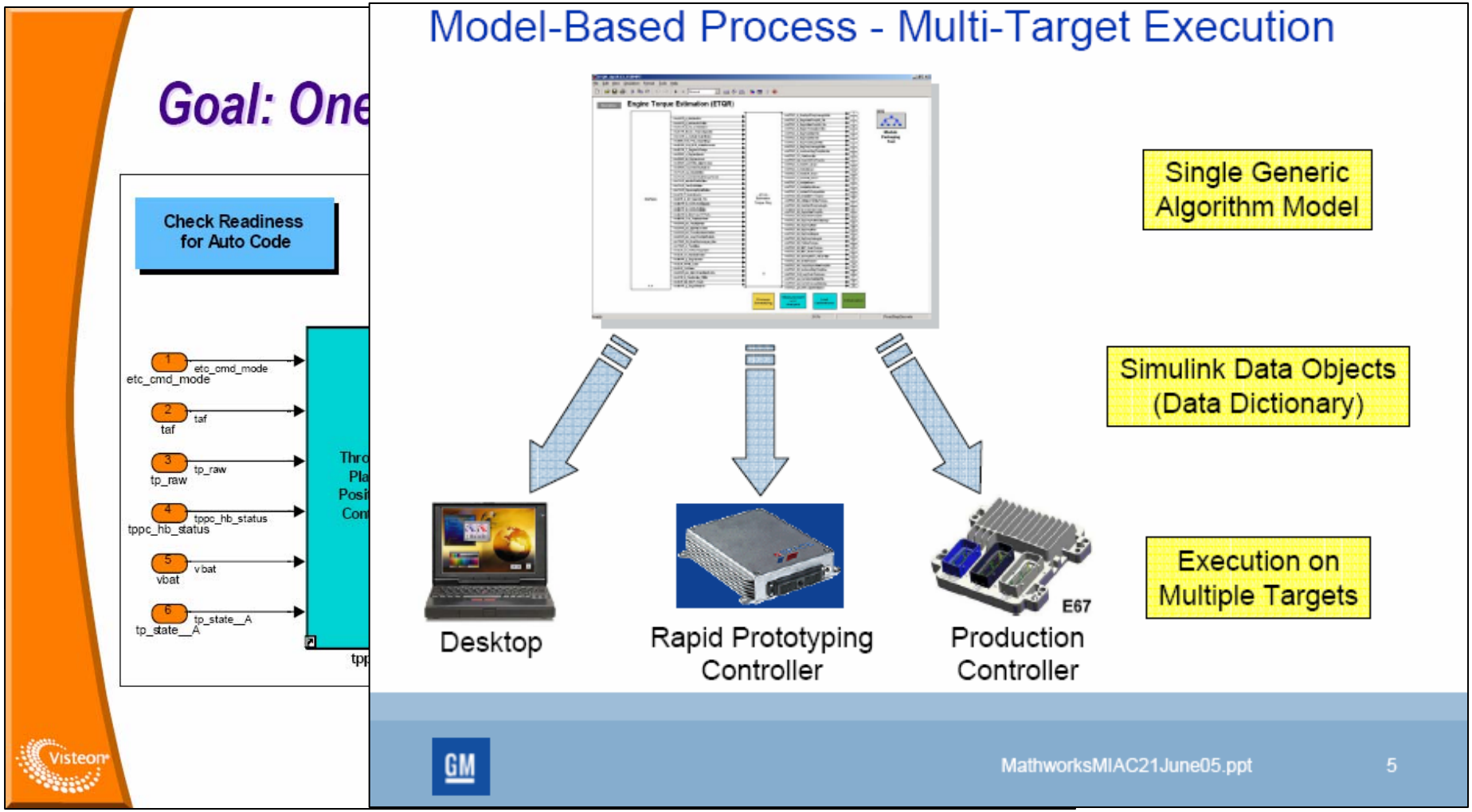
Keys to Success:

- Proficiency of users – requires mentoring by experts
- Standardized model style (especially tuned for code generation)
- Visteon custom tools

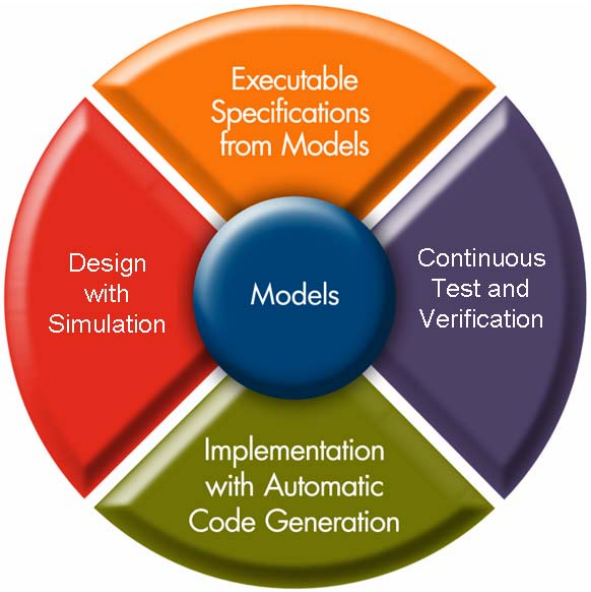
Visteon

4

MAC 2005 – Golden Model



Demo



- Executable specification
  - Multidomain
  - Rapid Simulation
  - Interaction options
- Detailed Design
  - Data Dictionary
- Code generation
  - RP/HIL
  - PCG
  - AUTOSAR
- Verification and Validation
  - Bidirectional trace
  - Model Advisor

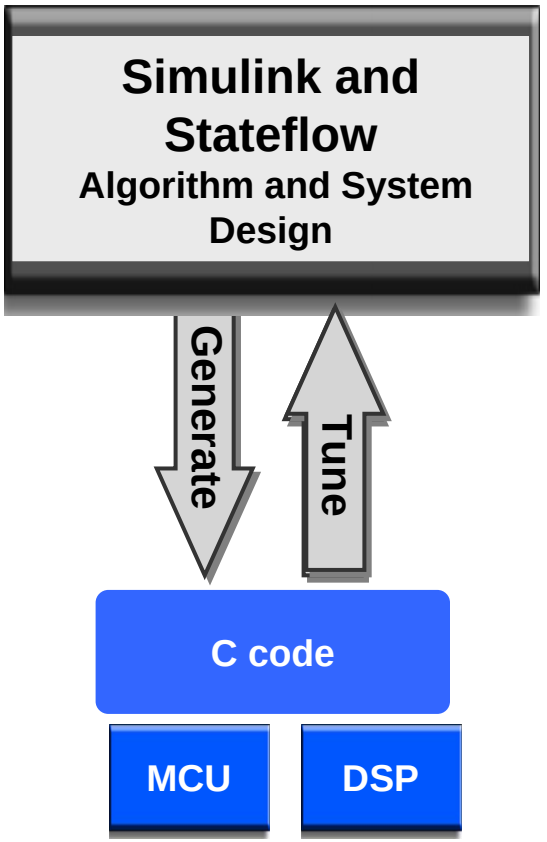
Code Generation – Early 2000s

Real-Time Workshop® Embedded Coder

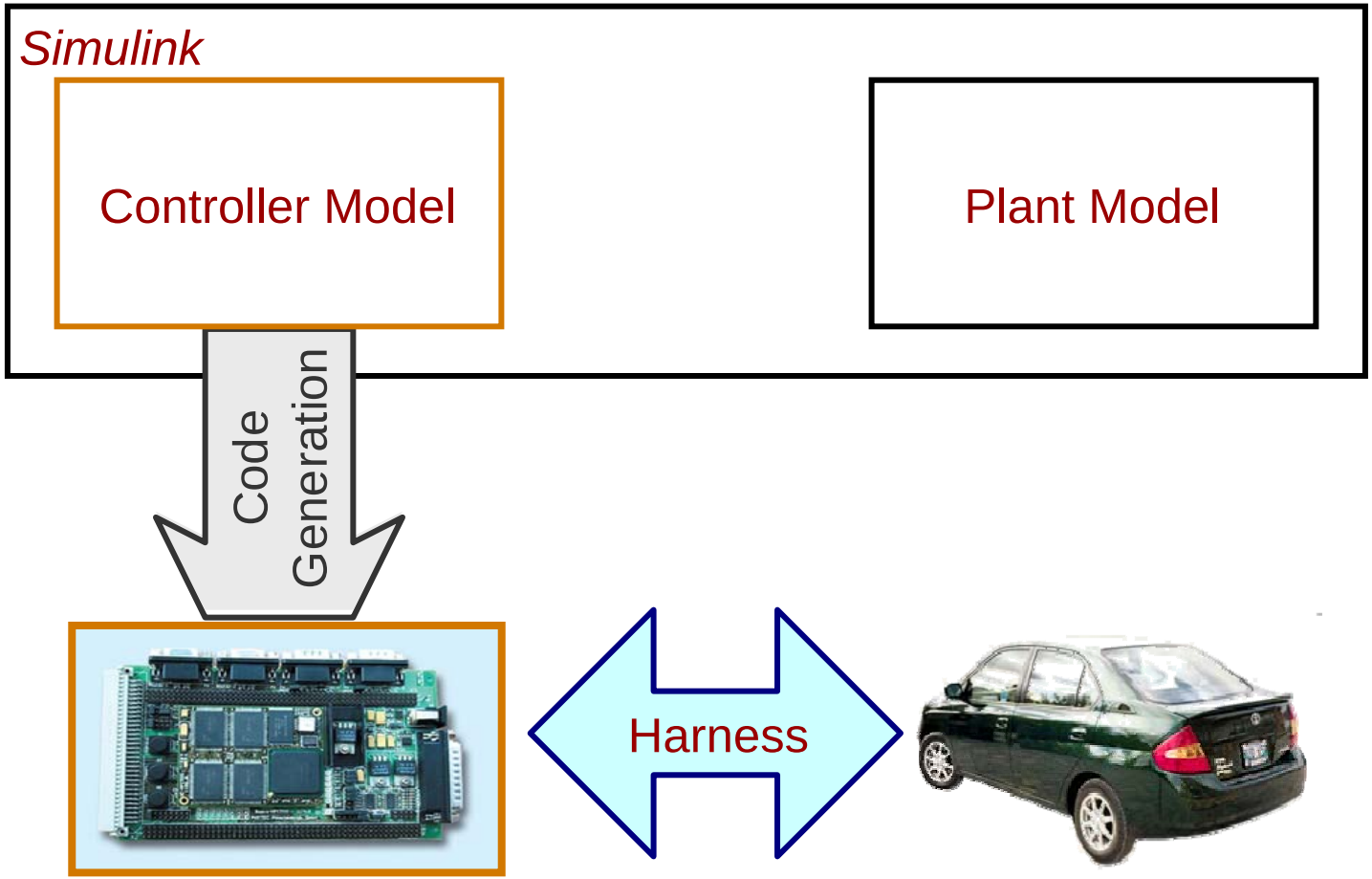
- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation

Embedded Targets

- Provide target specific blocks/features for
  - On-Target Rapid Prototyping

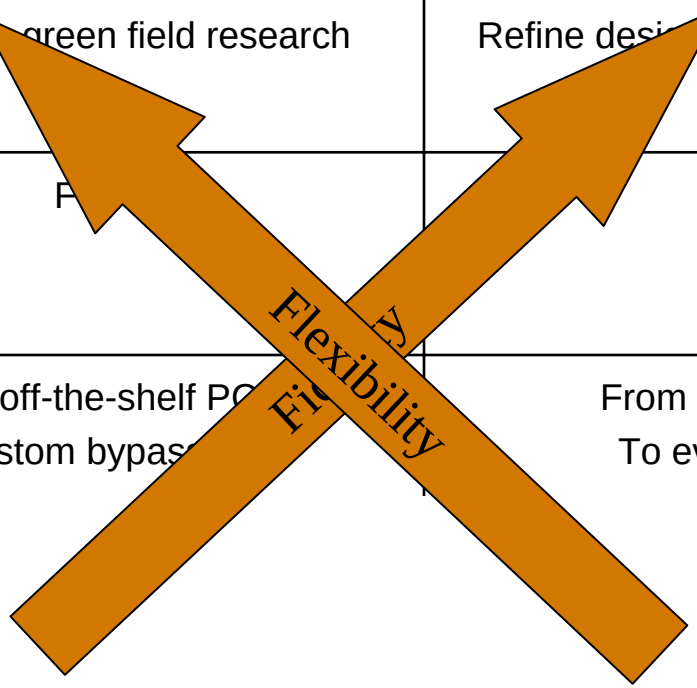


Rapid Prototyping



Rapid Prototyping Comparison

	Traditional Bypass	On-Target
Purpose	New ideas, green field research	Refine designs, production focused
Production Constraints (efficiency)	F	More
Hardware (Cost)	From off-the-shelf PC To custom bypas	From Existing ECUs To eval board HW



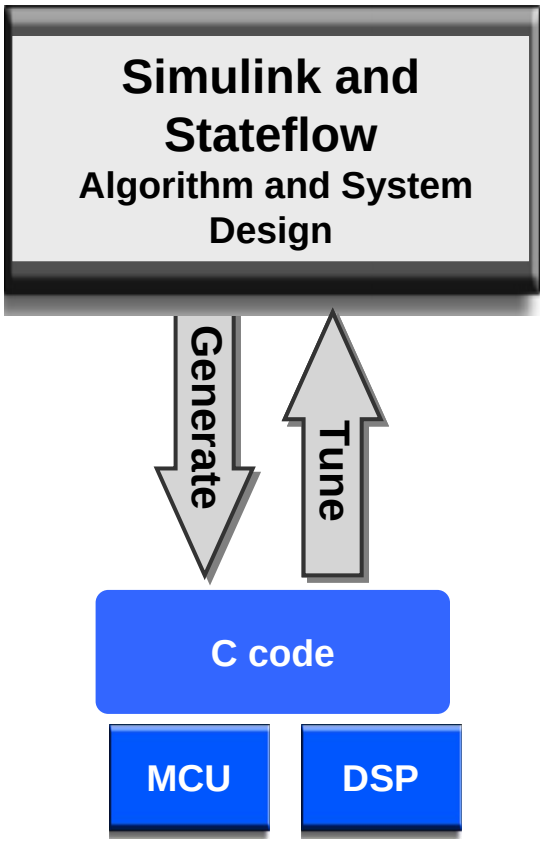
Code Generation – Early to Mid 2000s

Real-Time Workshop® Embedded Coder

- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation

Links and Targets

- Provide target specific blocks/features for
  - On-Target Rapid Prototyping
- SIL and PIL support



MAC 2004 – DC Trucks

Agenda

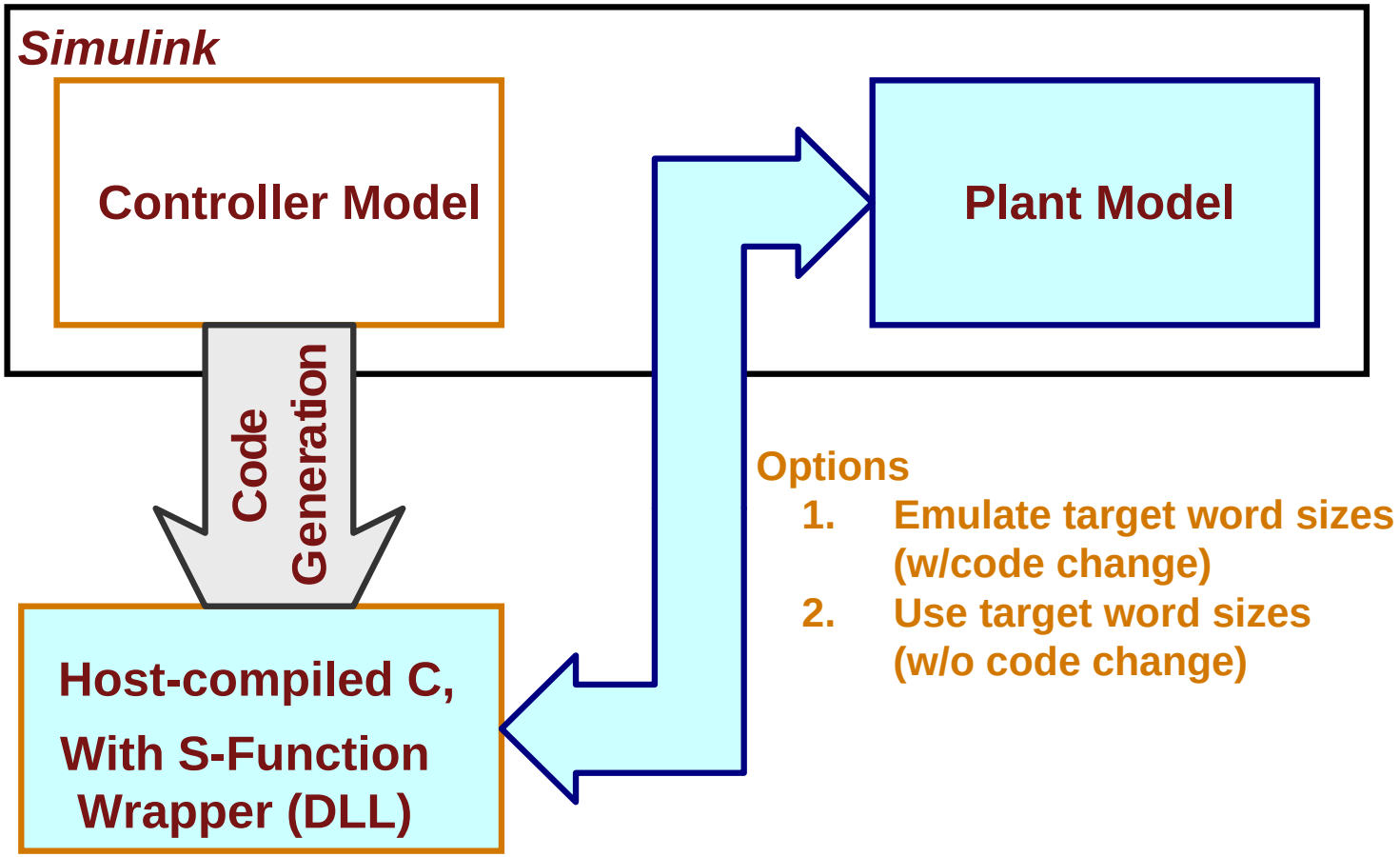
- Introduction
  - functional and technical overview
- Project description
  - motivation and development process
- C-Code analysis
  - analysing and documentation methods
- Control-module
  - Structure, metrics and co-operation methods
- Auto-Code generation
  - experiences and results
- Testing
  - SIL, PIL and HIL test methods

Conclusions

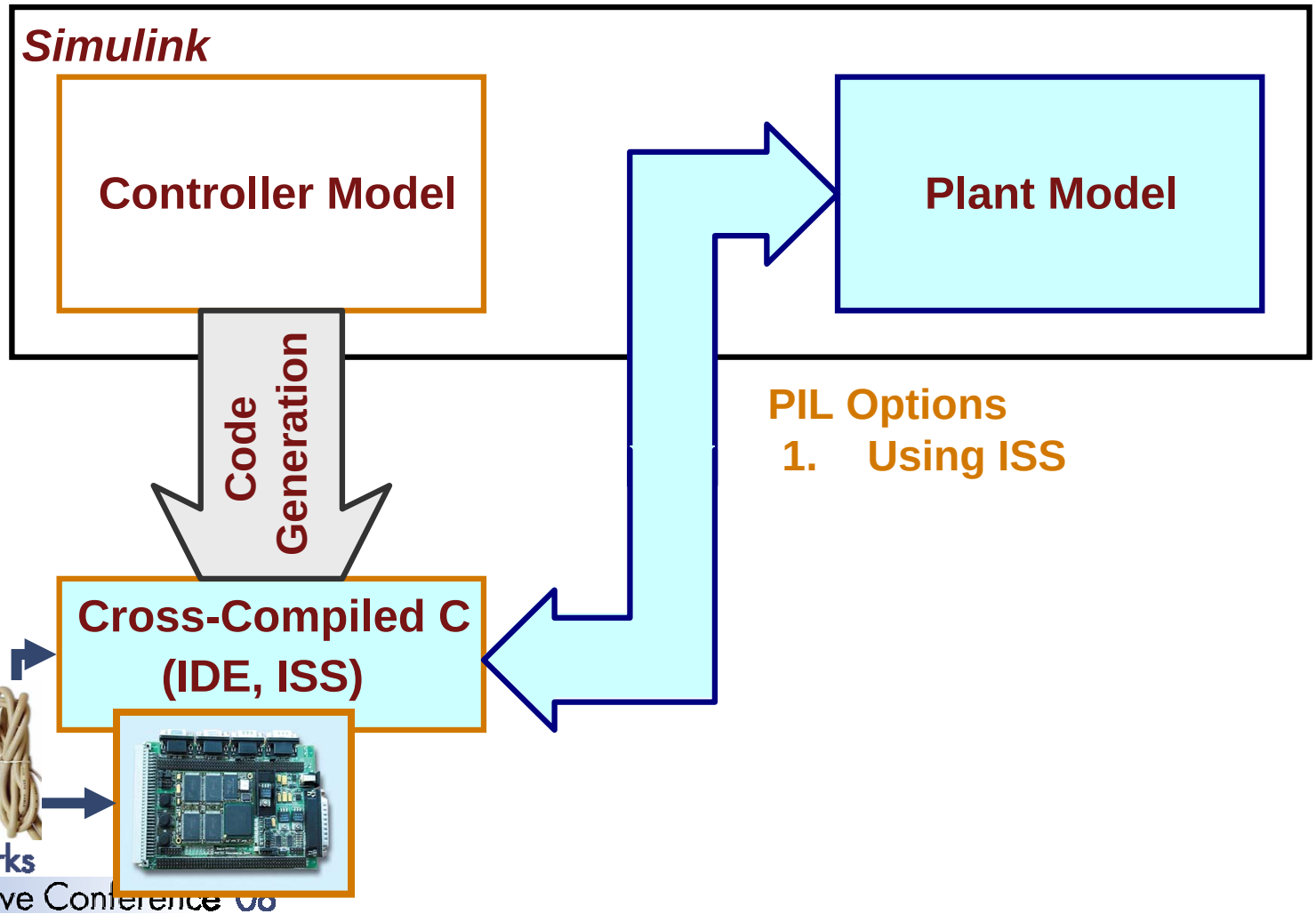
Results

- Project needs only 18 month until release
  - including analysis, restructuring, modelling and testing
- SIL based function development
  - high state of maturity before vehicle tests start
  - higher test efficiency
  - desktop debugging instead of debugging in vehicle
- Code generation
  - Embedded Coder meets our demands
  - code efficiency and readability like hand written code
- Project aims could be reached in time!

SIL

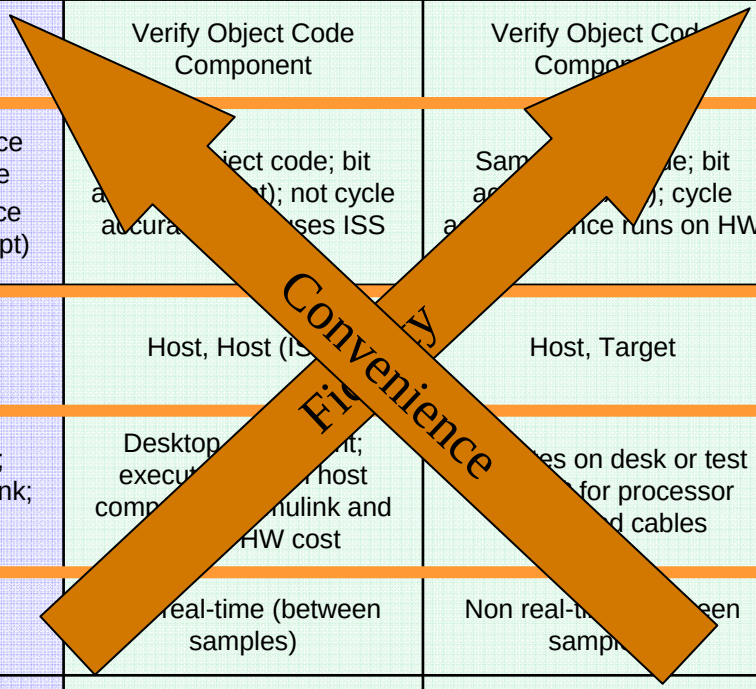

Simulink
Controller Model
Plant Model
Code Generation
Host-compiled C,
With S-Function
Wrapper (DLL)
Options
1. Emulate target word sizes (w/code change)
2. Use target word sizes (w/o code change)

PIL


Simulink
Controller Model
Plant Model
Code Generation
PIL Options
1. Using ISS
Cross-Compiled C
(IDE, ISS)

In-the-Loop Comparison

Description	SIL	PIL (ISS)	PIL (HW)	HIL
	Software-in-the-Loop	Processor-in-the-Loop		Hardware-in-the-Loop
Purpose	Verify Source Code Component	Verify Object Code Component	Verify Object Cod Compo	Verify Complete System Functionality
Fidelity	Emulated: same source code, not bit accurate Actual: Different source code, bit accurate (fix-pt)	iect code; bit a t); not cycle accura ses ISS	Sam e; bit ac); cycle a ice runs on HW	Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O
Platform (Plant, ECU)	Host, Host	Host, Host (IS	Host, Target	Real-Time System, Target
Convenience	Desktop convenient; executes just in Simulink; no HW cost	Desktop execut nt; comm host nulink and HW cost	es on desk or test for processor d cables	Executes in test bench or lab; $$ for processor, ECU, I/O, cables
Real-Time	Non real-time	eal-time (between samples)	Non real-t een sampl	Hard real-time
Engineers	Systems or Software Engineers	Software or Test Engineers	Software or Test Engineers	Systems or Test Engineers



Link and Target Products – R2007a

Links

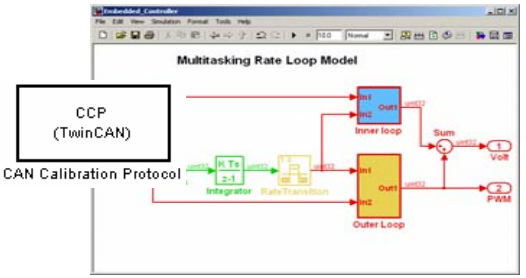
- PIL via IDEs
  - Altium TASKING®
  - Analog Devices VisualDSP++®
  - TI's Code Composer Studio™
  - Mentor Graphics ModelSim®
  - Cadence® Incisive®
- Project creation
- Optimizations

Targets

- Add-ons to Links
- Device driver blocks
- Optimizations

Many third party offerings

Simulink®



TASKING

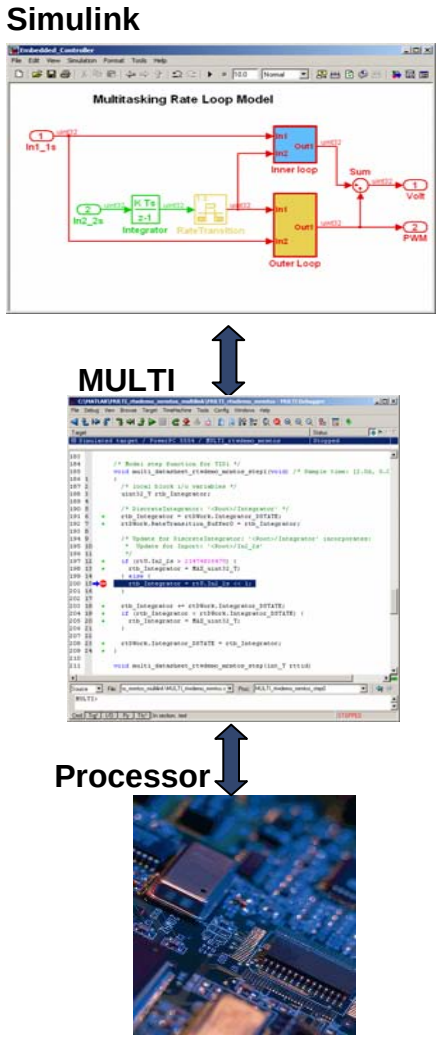
ST10

Tricore

Embedded IDE Link™ MU - R2007b+

(for Green Hills® MULTI®)

- Automates build and test of embedded code using MATLAB®, Simulink® and MULTI® (from Green Hills®)
- For systems and software engineers who want to deploy and verify automatically generated code quickly
- Supports key processors including:
  - Freescale™ MPC5554 and MPC7447
  - NEC V850
  - Analog Devices® Blackfin®



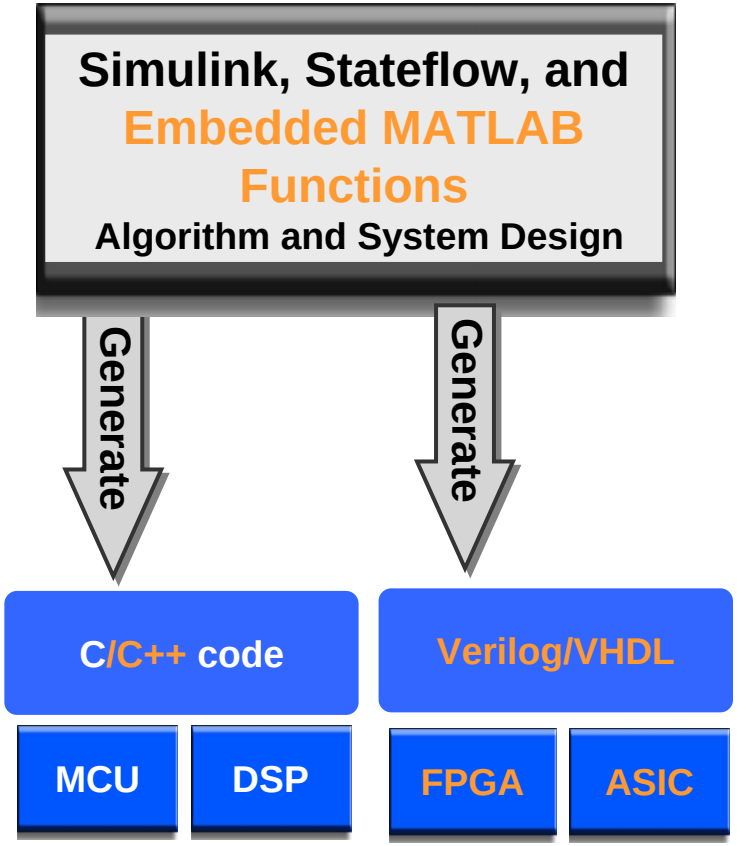
Code Generation – Mid 2000s

Real-Time Workshop Embedded Coder

- C and C++ Production Code Generation

Simulink® HDL Coder

- Verilog and VHDL Code Generation



You can deploy code on any microprocessor or hardware device.

Code Verification – Mid 2000s

Real-Time Workshop Embedded Coder

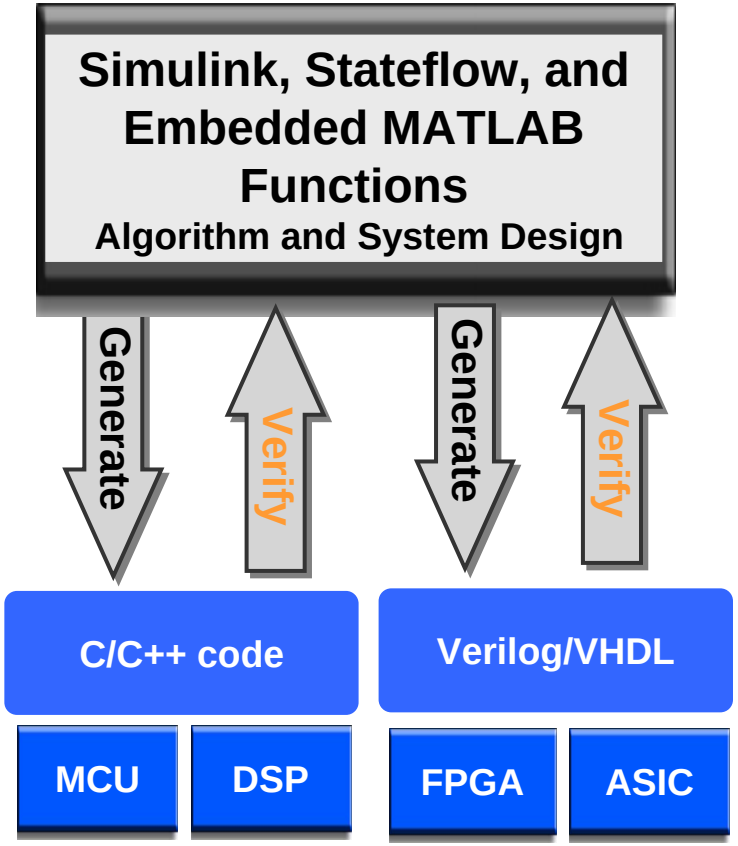
- C and C++ Production Code Generation

Simulink® HDL Coder

- Verilog and VHDL Code Generation

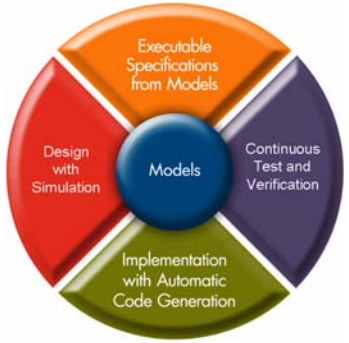
Links and Targets

- On-Target Prototyping
- SIL and PIL



You can verify code on any microprocessor or hardware device.

Demo



- PIL Testing

Simulation is not the only way to do Verification and Validation

Introduced in 2007…

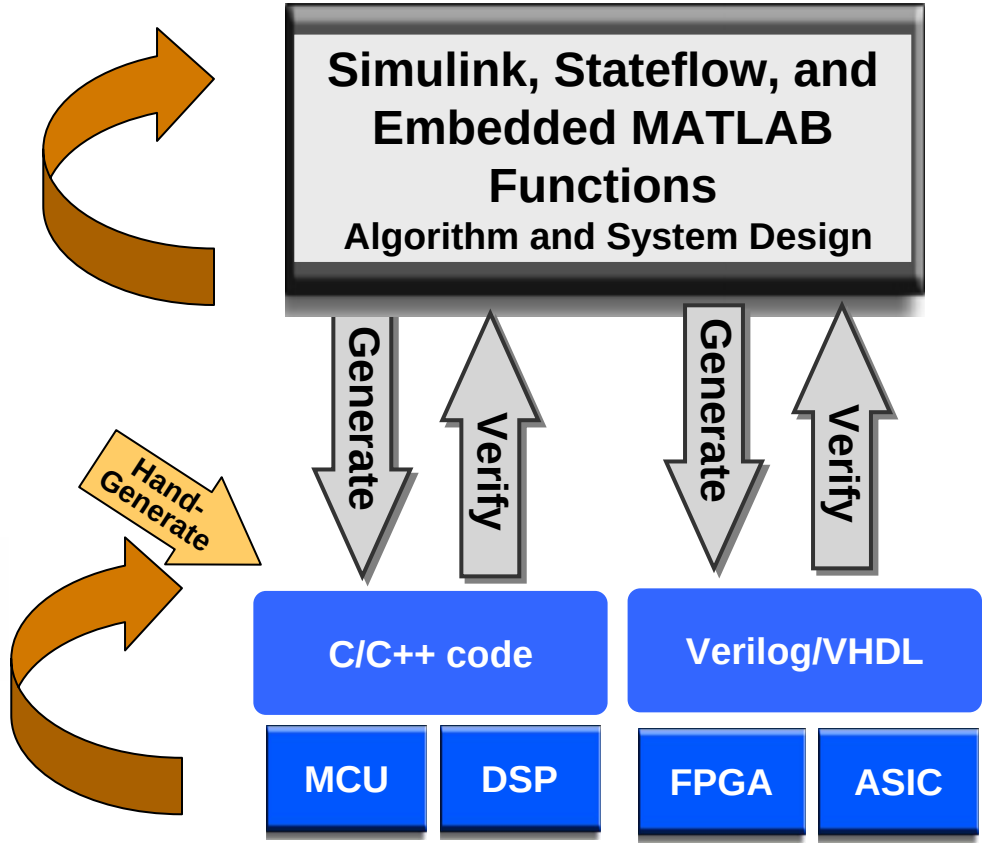
Simulink Design Verifier

Generate tests and prove model properties using formal methods

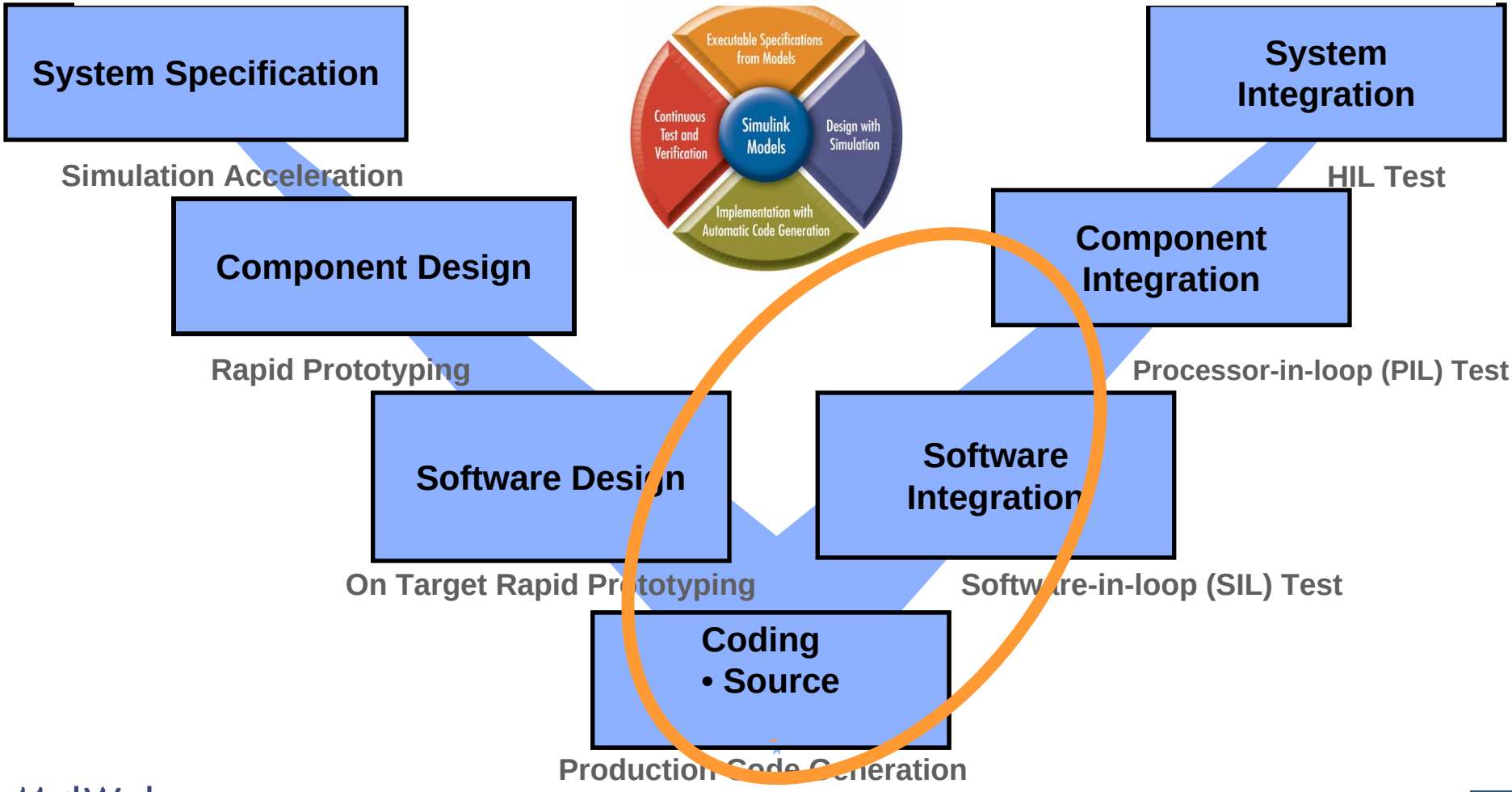
prover plugged in

The MathWorks Acquires PolySpace Technologies, Leading Developer Of Embedded System Code Verification Tools

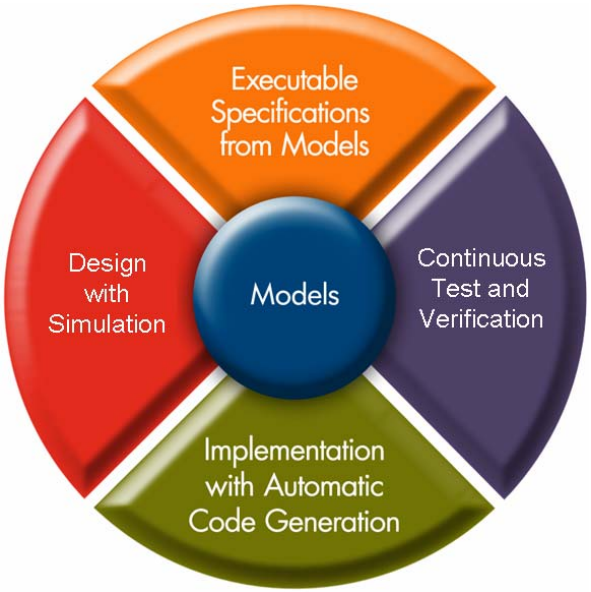
PolySpace TECHNOLOGIES
Delivering safe code - faster



ECU Development – Today (R2008a)



Agenda



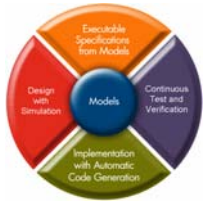
PCG Historical Review

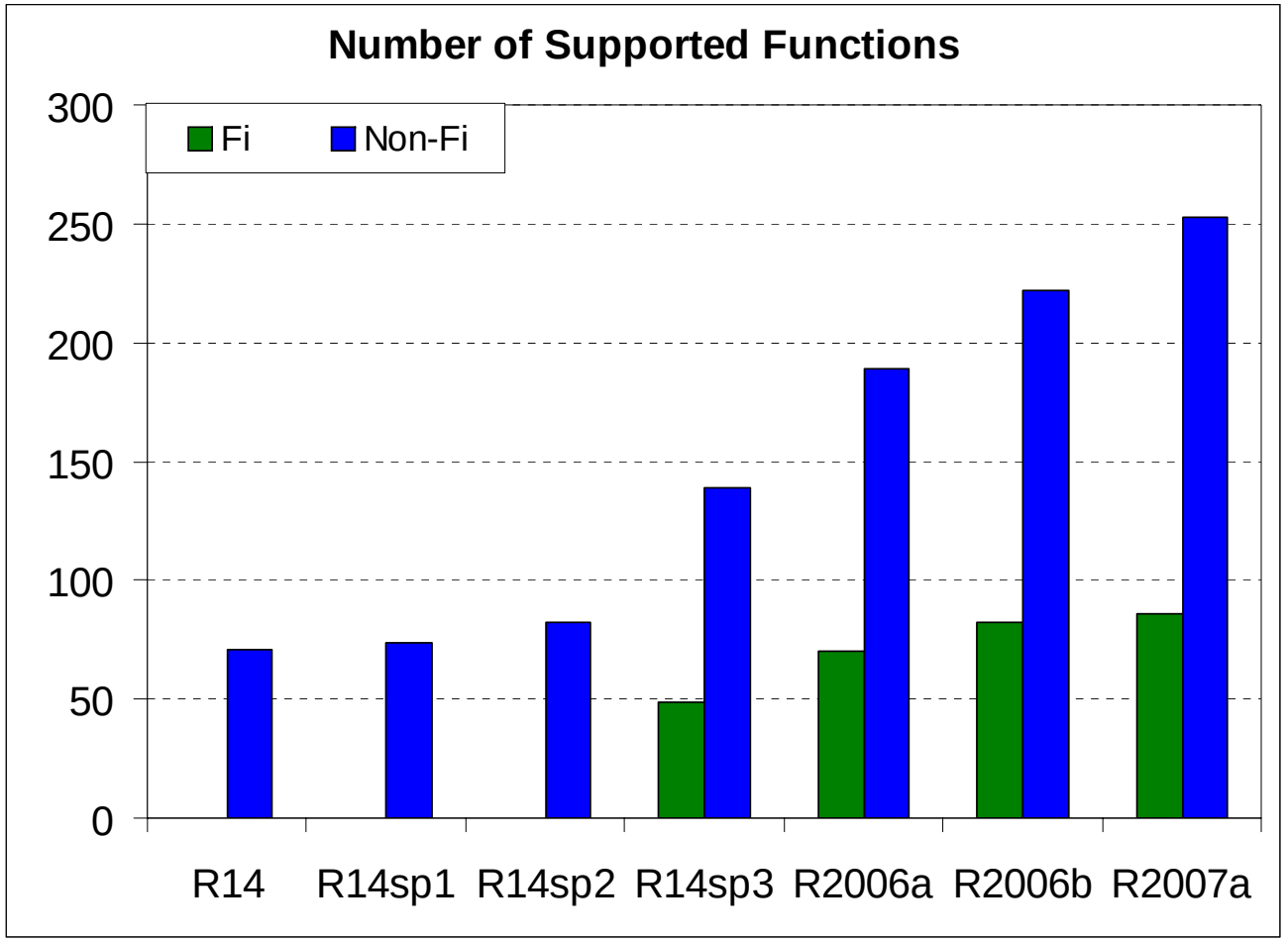
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- **Today (R2007a to 2008a)**

New features and trends

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification

Embedded MATLAB Functions - 2007a

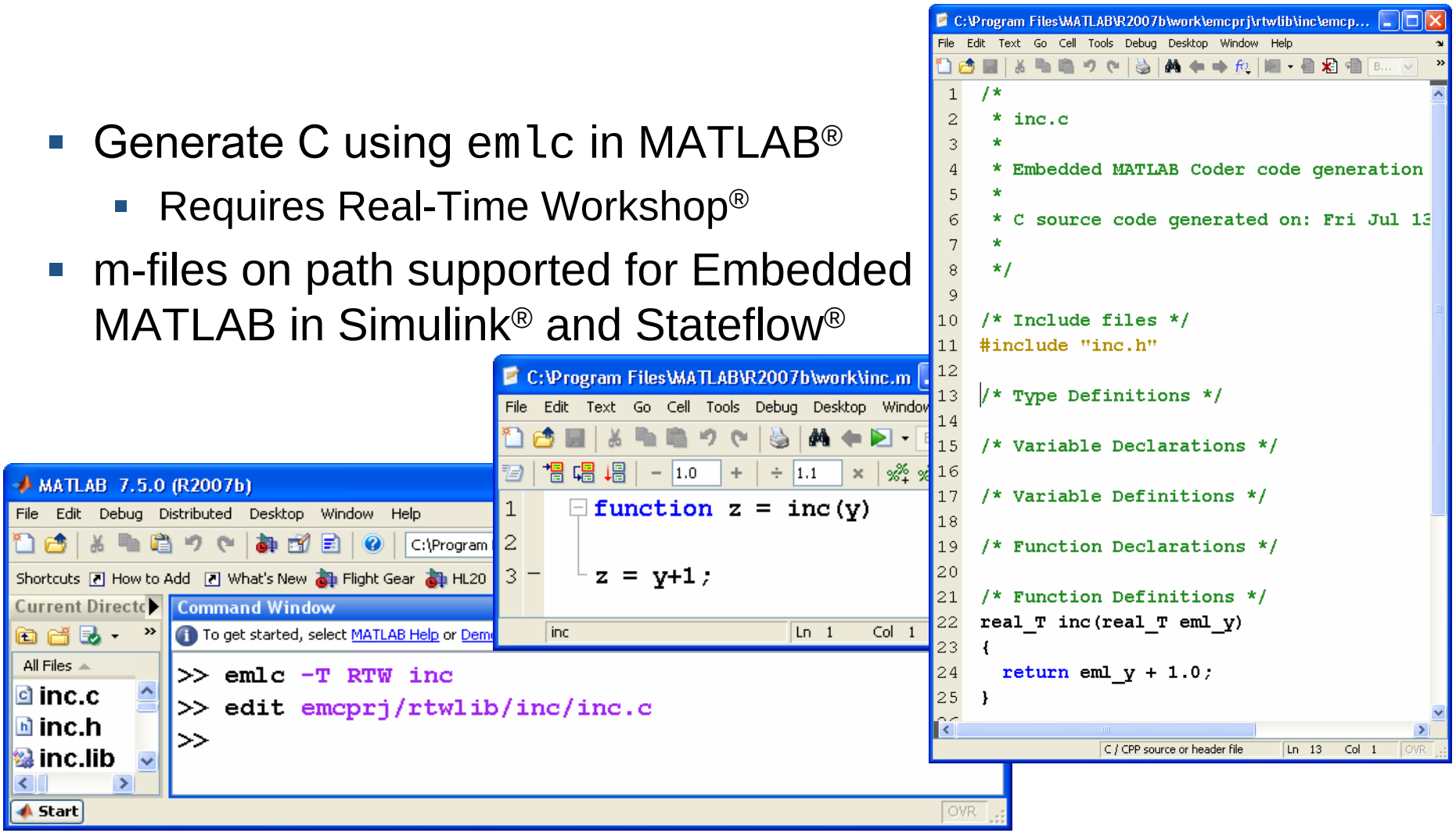




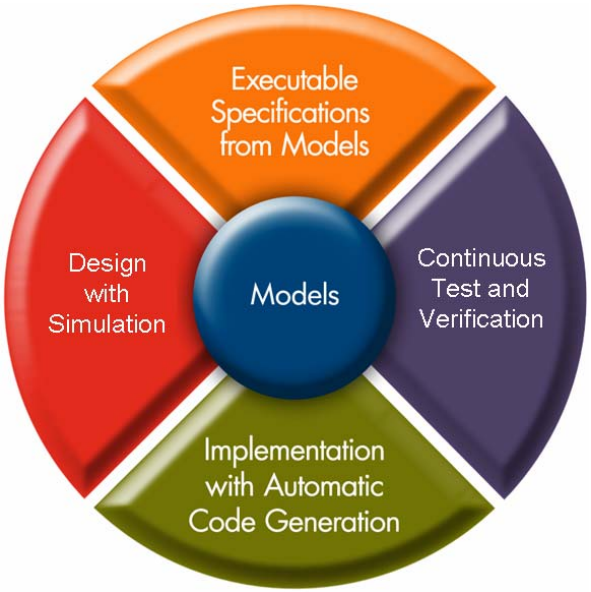
>>eml_frames, eml_nd

Embedded MATLAB™ Improvements – R2007b

- Generate C using `emlc` in MATLAB®
  - Requires Real-Time Workshop®
- m-files on path supported for Embedded MATLAB in Simulink® and Stateflow®



Agenda



PCG Historical Review

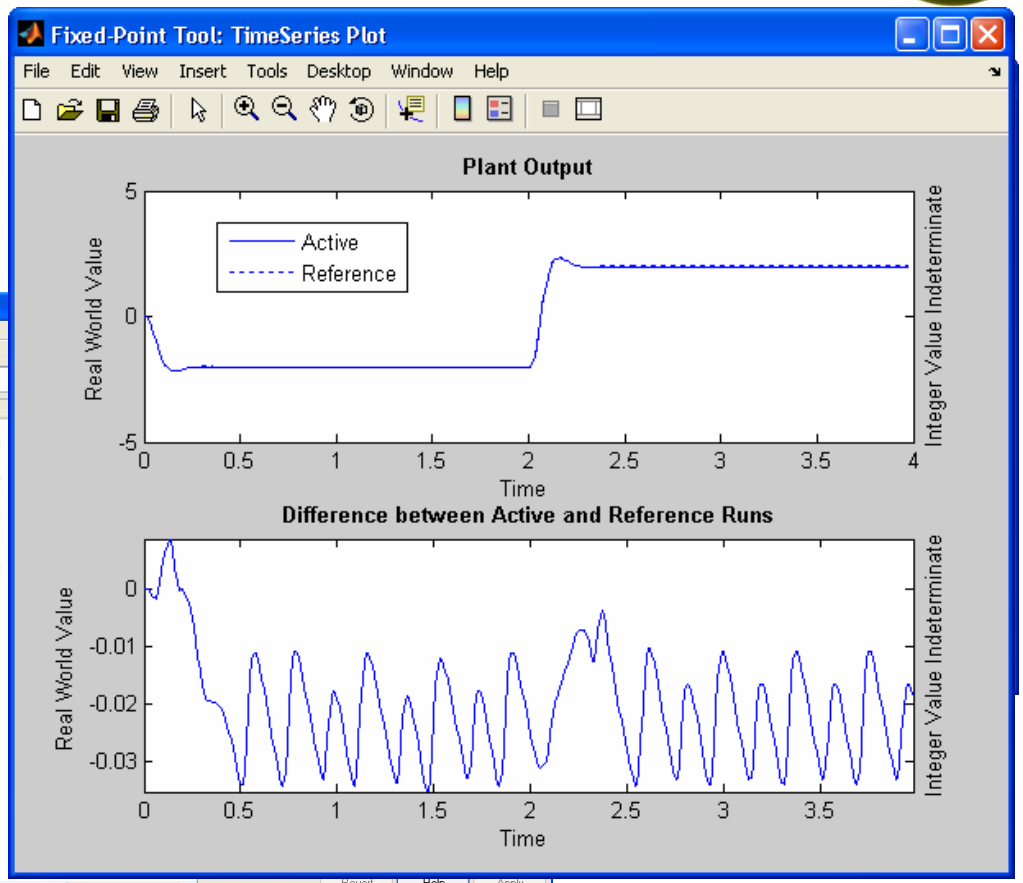
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

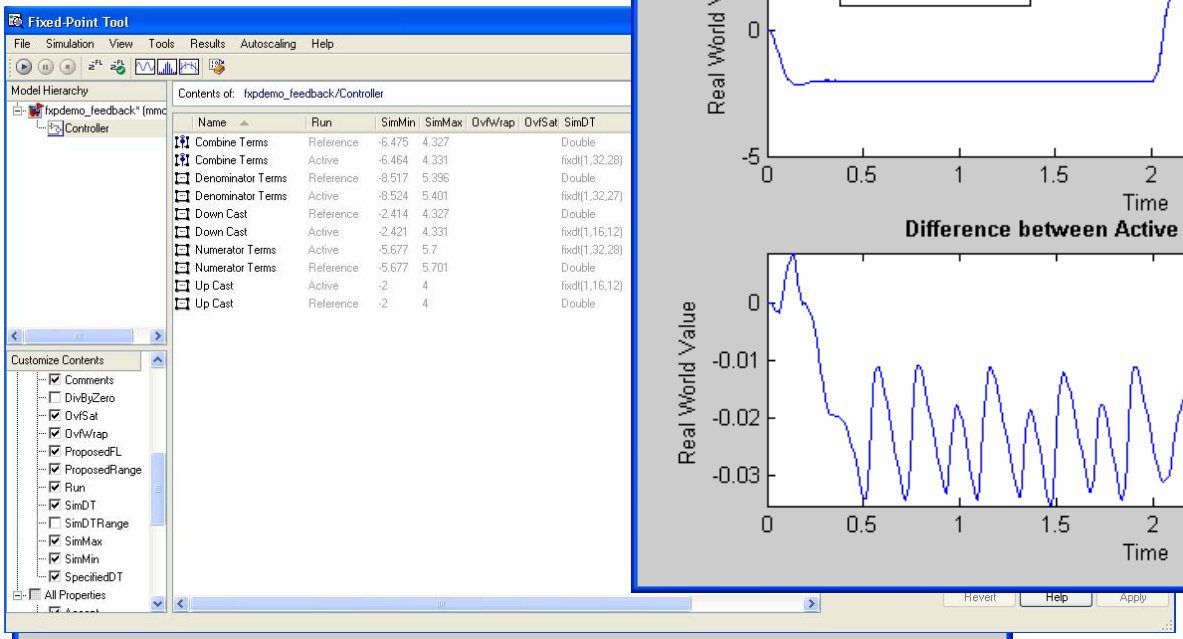
New features

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification

Fixed Point Tool – 2007a

- Includes:
  - Data type override
  - Automated scaling
  - Over/under flow detection
  - Fixed vs. float plots





>>fxpdemo_feedback

Fixed-Point Tool - R2008a

Design Min/Max

Problem

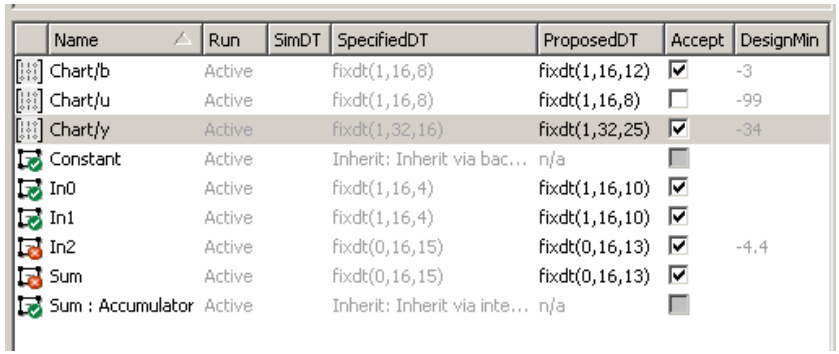
- Fixed-Point Tool scaling based on design min and max data ranges was not supported

Solution

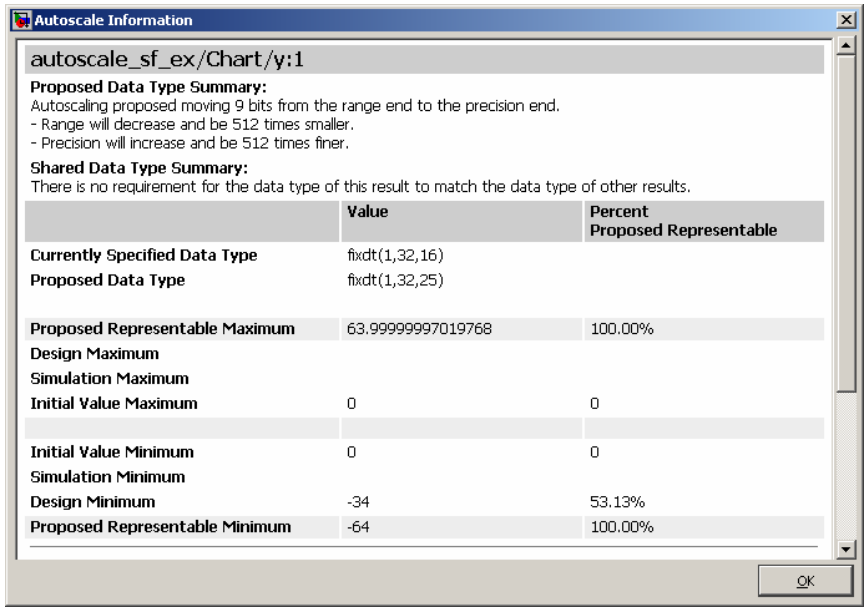
- Fixed-Point Tool now leverages design minimum and maximum from blocks and Stateflow data

Benefit

- Better fits user workflow
- Provides autoscaling that does not require simulation



Name	Run	SimDT	SpecifiedDT	ProposedDT	Accept	DesignMin
Chart/b	Active		fixdt(1,16,8)	fixdt(1,16,12)	☑	-3
Chart/u	Active		fixdt(1,16,8)	fixdt(1,16,8)	☐	-99
Chart/y	Active		fixdt(1,32,16)	fixdt(1,32,25)	☑	-34
Constant	Active		Inherit: Inherit via bac...	n/a	☐	
In0	Active		fixdt(1,16,4)	fixdt(1,16,10)	☑	
In1	Active		fixdt(1,16,4)	fixdt(1,16,10)	☑	
In2	Active		fixdt(0,16,15)	fixdt(0,16,13)	☑	-4.4
Sum	Active		fixdt(0,16,15)	fixdt(0,16,13)	☑	
Sum : Accumulator	Active		Inherit: Inherit via inte...	n/a	☐	



Autoscale Information

autoscale_sf_ex/Chart/y:1

Proposed Data Type Summary:
Autoscaling proposed moving 9 bits from the range end to the precision end.
- Range will decrease and be 512 times smaller.
- Precision will increase and be 512 times finer.

Shared Data Type Summary:
There is no requirement for the data type of this result to match the data type of other results.

	Value	Percent Proposed Representable
Currently Specified Data Type	fixdt(1,32,16)	
Proposed Data Type	fixdt(1,32,25)	
Proposed Representable Maximum	63.99999997019768	100.00%
Design Maximum		
Simulation Maximum		
Initial Value Maximum	0	0
Initial Value Minimum	0	0
Simulation Minimum		
Design Minimum	-34	53.13%
Proposed Representable Minimum	-64	100.00%

OK

Fixed-Point Advisor – R2008a

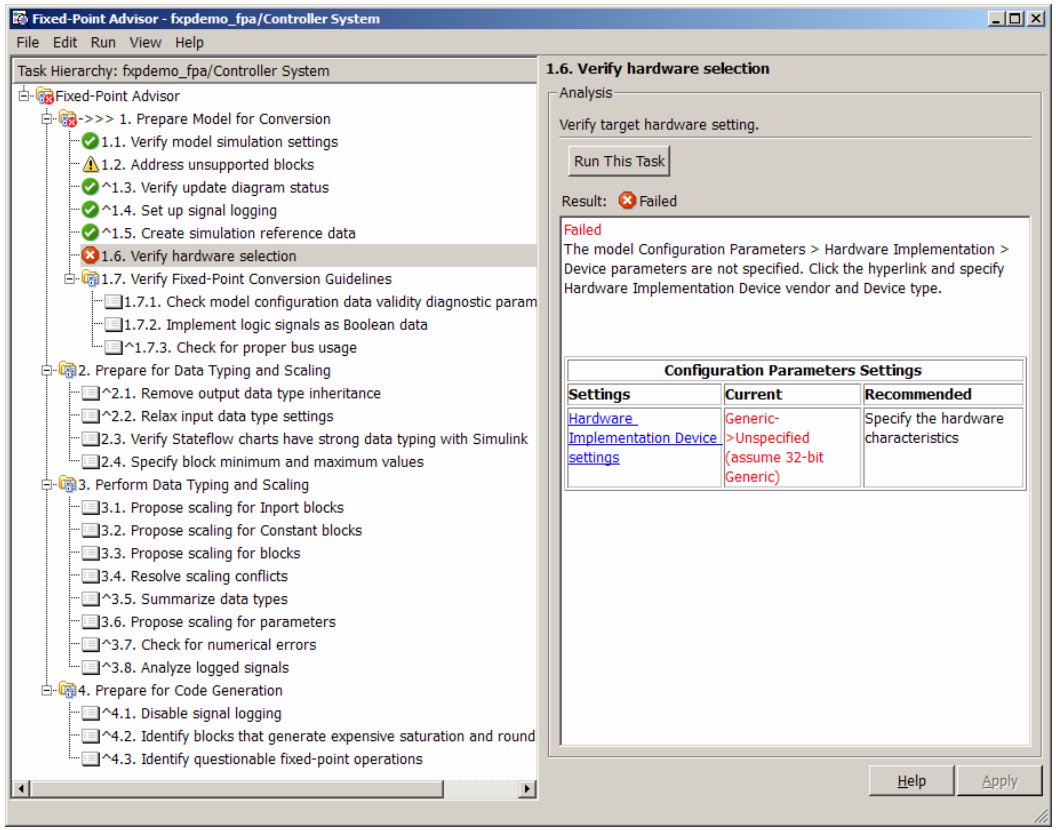
Float- to fixed-point conversion

Problem

- Obtain an initial float- to fixed-point conversion with least effort

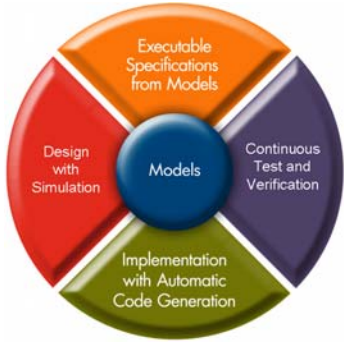
Solution

- Fixed-Point Advisor helps
  - Set model parameters
  - Set block parameters
  - Perform fixed-point conversion
  - Validate conversion using floating point results
  - Prepare for code generation
- Complement Fixed-Point Tools, which optimizes fixed-point scaling


Fixed-Point Advisor - fxpdemo_fpa/Controller System
File Edit Run View Help
Task Hierarchy: fxpdemo_fpa/Controller System
Fixed-Point Advisor
->>> 1. Prepare Model for Conversion
1.1. Verify model simulation settings
1.2. Address unsupported blocks
^1.3. Verify update diagram status
^1.4. Set up signal logging
^1.5. Create simulation reference data
1.6. Verify hardware selection
1.7. Verify Fixed-Point Conversion Guidelines
1.7.1. Check model configuration data validity diagnostic param
1.7.2. Implement logic signals as Boolean data
^1.7.3. Check for proper bus usage
2. Prepare for Data Typing and Scaling
^2.1. Remove output data type inheritance
^2.2. Relax input data type settings
2.3. Verify Stateflow charts have strong data typing with Simulink
2.4. Specify block minimum and maximum values
3. Perform Data Typing and Scaling
3.1. Propose scaling for Inport blocks
3.2. Propose scaling for Constant blocks
3.3. Propose scaling for blocks
3.4. Resolve scaling conflicts
^3.5. Summarize data types
3.6. Propose scaling for parameters
^3.7. Check for numerical errors
^3.8. Analyze logged signals
4. Prepare for Code Generation
^4.1. Disable signal logging
^4.2. Identify blocks that generate expensive saturation and round
^4.3. Identify questionable fixed-point operations
1.6. Verify hardware selection
Analysis
Verify target hardware setting.
Run This Task
Result: Failed
Failed
The model Configuration Parameters > Hardware Implementation > Device parameters are not specified. Click the hyperlink and specify Hardware Implementation Device vendor and Device type.
Configuration Parameters Settings
Settings
Current
Recommended
Hardware Implementation Device settings
Generic->Unspecified (assume 32-bit Generic)
Specify the hardware characteristics
Help
Apply

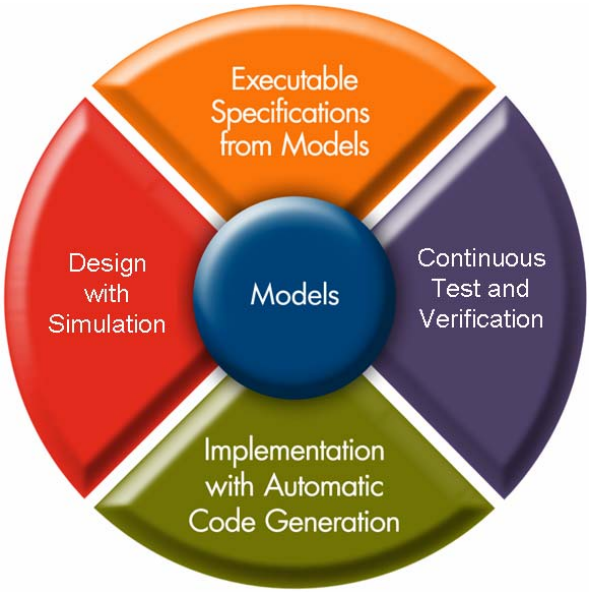
>>fxpdemo_fpa

Demo



- Fixed Point Advisor

Agenda



PCG Historical Review

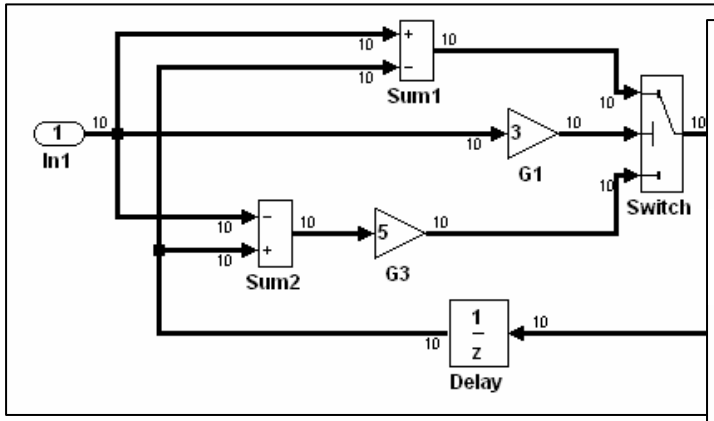
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification

Code Efficiency - R2007a

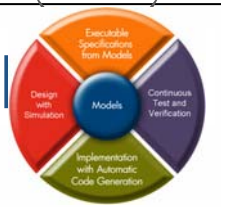
Wide Signals (for-loops)



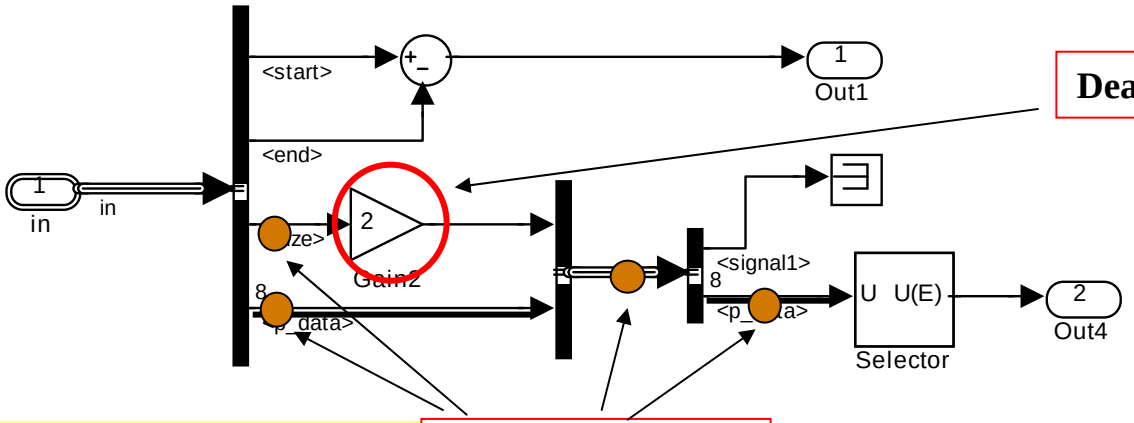

```
void rtwdemo_forloop_R2007a_step(void)
{
  real_T tmp
  for                    ++) {
                         ] >= 0.0) {
                         - rtDWork.Delay_DSTATE[i];
    } else {
      tmp = (rtDWork.Delay_DSTATE[i] - rtU.In1[i]) * 5.0;
    }

    rtY.Out1[i] = tmp;
    rtDWork.Delay_DSTATE[i] = tmp;
  }
}
```

```
void rtwdemo_forloop_R14SP2_step(void)
{
  real_T rtb_S
  real_T rtb_Switch[10]

  {
    int32_T i1;

  for ; i1<10; i1++) {
        emo_forloop_R14SP2_U.In1[i1] * 3.0 >= 0.0) {
      rtb_Switch[i1] = rtwdemo_forloop_R14SP2_U.In1[i1] -
        rtwdemo_forloop_R14SP2_DWork.Delay_DSTATE[i1];
    } else {
      rtb_Switch[i1] = (rtwdemo_forloop_R14SP2_DWork.Delay_DSTATE[i1] -
        rtwdemo_forloop_R14SP2_U.In1[i1]) * 5.0;

    rloop_R14SP2_Y.Out1[i1] = rtb_Switch[i1];

    rloop_R14SP2_DWork.Delay_DSTATE[i1] = rtb_Switch[i1];
  }
```

```
wdemo_forloop_R13SP2_step(void)

 rtb_Sw  real_T rtb_Switch[

{
  int_T i1;

  y0 = &rtb_Switch[0];

= 0.0) {

STATE[i1]);

] +
STATE[i1]) * 5.0);

t1[0];

  int_T i1;

  const real_T *u0 = &rtb_Switch[0];
  real_T *dw_DSTATE = &rtwdemo_forloop_R13SP2_DWork.Delay_DSTATE[0];

for =0; i1 < 10; i1++) {
  _DSTATE[i1] = u0[i1];
```



Improved Bus Code – R2008a



R2007b

```
void s_step(void) {
  bus_2 rtb_BusCreator;
  int32_T rtb_size;
  int32_T rtb_signal1;
  uint8_T rtb_p_data[8];
  rtb_size = s_U.in.size;
  (void) memcpy(&(rtb_p_data[0]),&s_U.in.p_data, 8*sizeof(u
  s_Y.Out1 = s_U.in.start - s_U.in.end;
  rtb_signal1 = rtb_size << 1U;
  rtb_BusCreator.size = rtb_signal1;
  (void) memcpy(&(rtb_BusCreator.p_data),rtb_p_data, 8*size
  (void) memcpy(&(rtb_p_data[0]),&rtb_BusCreator.p_data, 8*sizeof(uint8_T));
  s_Y.Out4 = rtb_p_data[2];
}
```

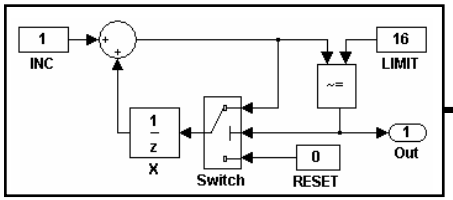
```
void s_step(void) {
  s_Y.Out1 = s_U.in.start - s_U.in.end;
  s_Y.Out4 = s_U.in.p_data[2];
}
```

R2008a

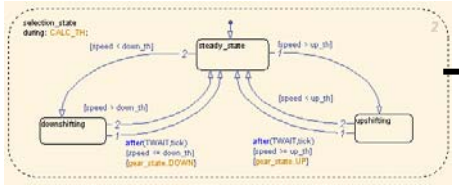
Improving bus code efficiency was a top request for this release

Coder Infrastructure Circa 1999

IR – Intermediate Representation
TLC – Template Language Compiler



Simulink



Stateflow

MATLAB

Simulink IR

Stateflow IR

Deployment Support Only

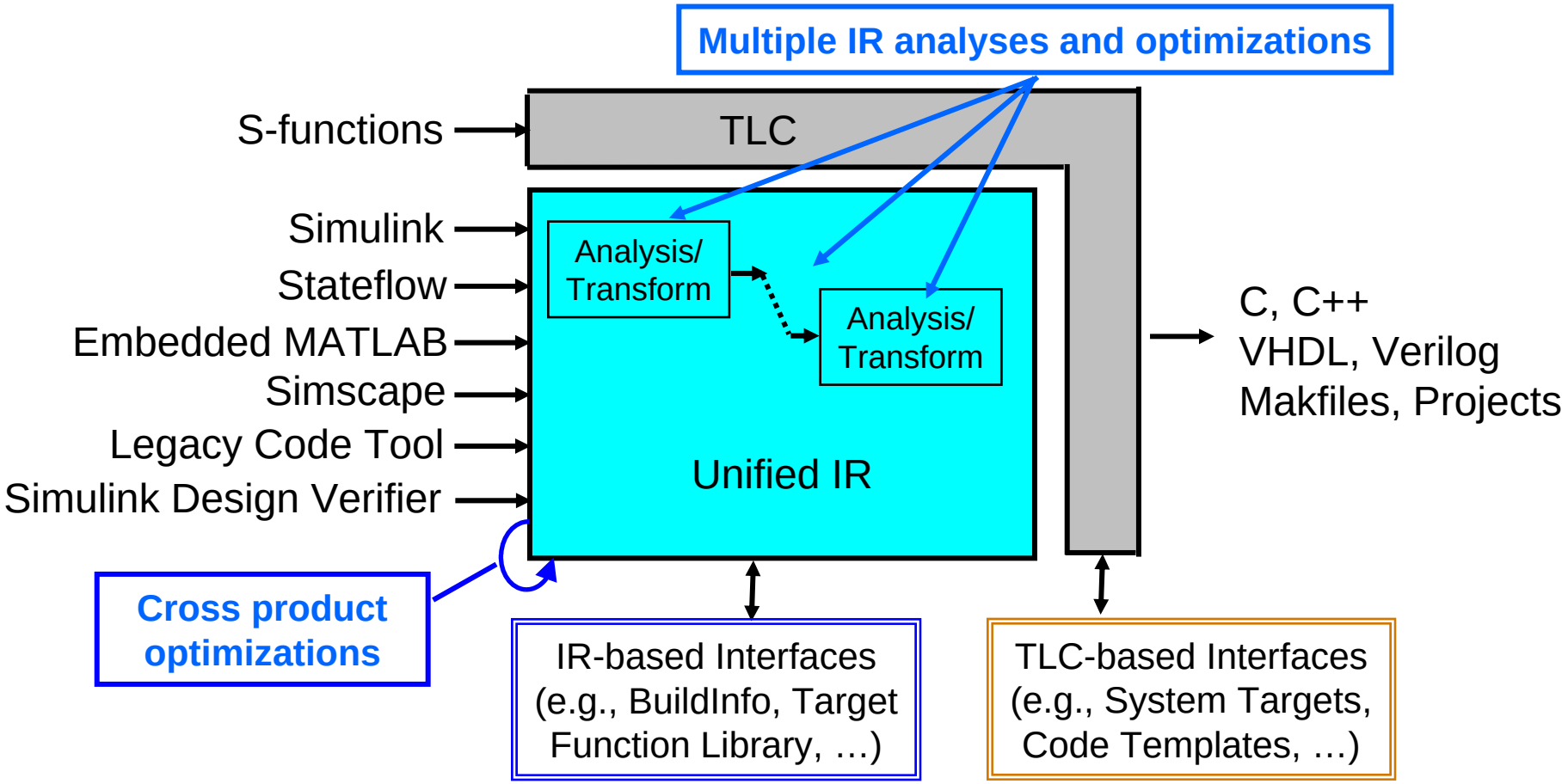
TLC

C, makefiles

TLC-based Interfaces (S-Functions, System Targets (e.g. GRT, ERT))

Coder Infrastructure 2008

- Unified for all domains, highly efficient, extensible

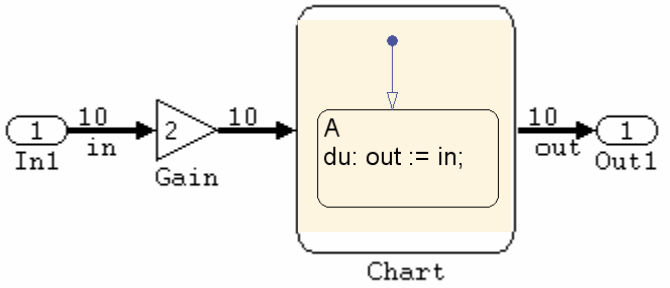


Cross Product Optimizations – R2008a

- Optimizations are now enabled across code segments generated for Simulink and Stateflow

R2007b

```
real_T rtb_Gain[10];
…
for (i = 0; i < 10; i++) {
  rtb_Gain[i] = 2.0 * in[i];
}
if (cu_DWork.is_active_c1_cu == 0) {
  …
} else {
    for (sf_i1 = 0; sf_i1 < 10; sf_i1++) {
      out[sf_i1] = rtb_Gain[sf_i1];
    }
}
```



R2008a

```
if (cu_DWork.is_active_c1_cu == 0) {
  …
} else {
  for (i = 0; i < 10; i++) {
    out[i] = 2.0 * in[i];
  }
}
```

Target Specific Optimizations – R2007b

Example Results

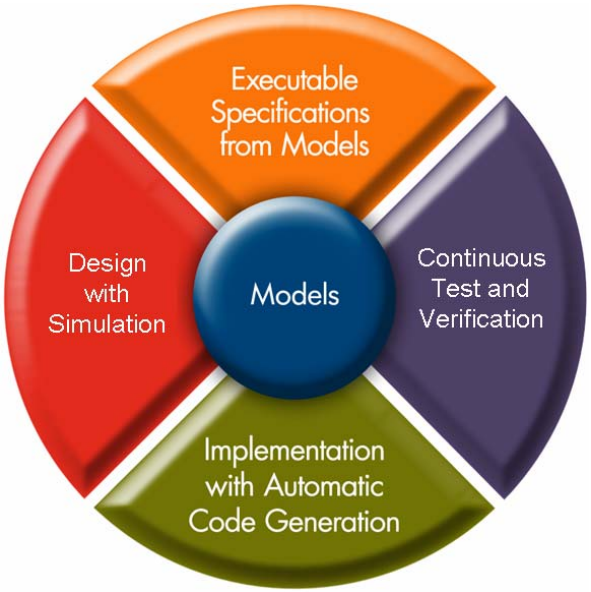
Processor	Algorithm Function	Target Specific Generated Code*	Speedup (over ANSI-C)

Tricore® By Infineon®	Fixed-pt Math (int32, int32, saturated)	int32_T tricore_add_s32_s32_s32_sat(int32_T a, int32_T b)
		{
		return (__sat int)a + b;
		}
			17.1X

C6416™ DSP By TI	FIR 128-sample (int32, 50-taps, saturated)	int32_T TI_C6000_s32_add_s32_s32_sat(int32_T a, int32_T b)
		{
		return _sadd(a, b);
		}
			6.2X

***Implemented using Target Function Libraries**

Agenda



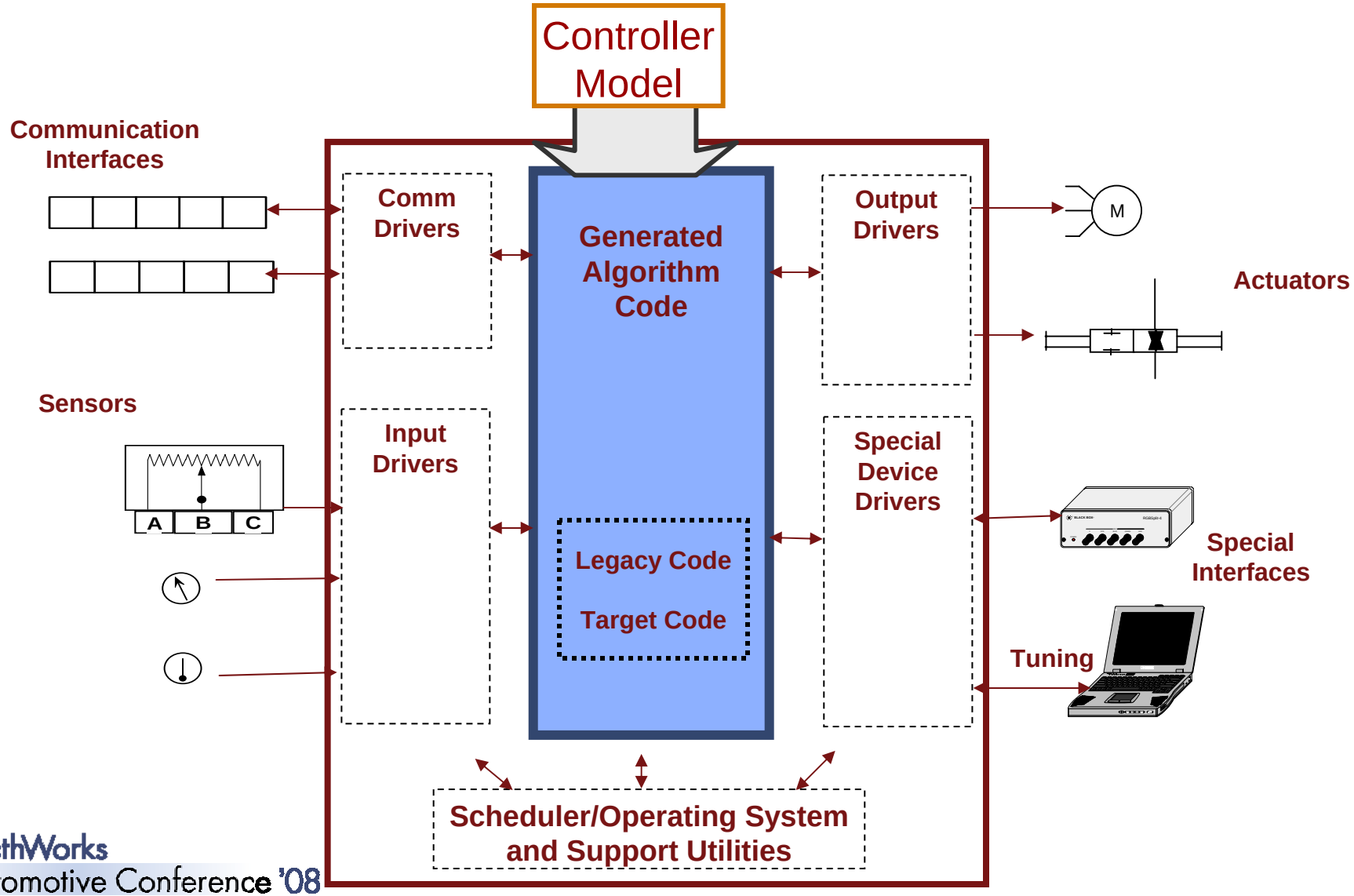
PCG Historical Review

- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

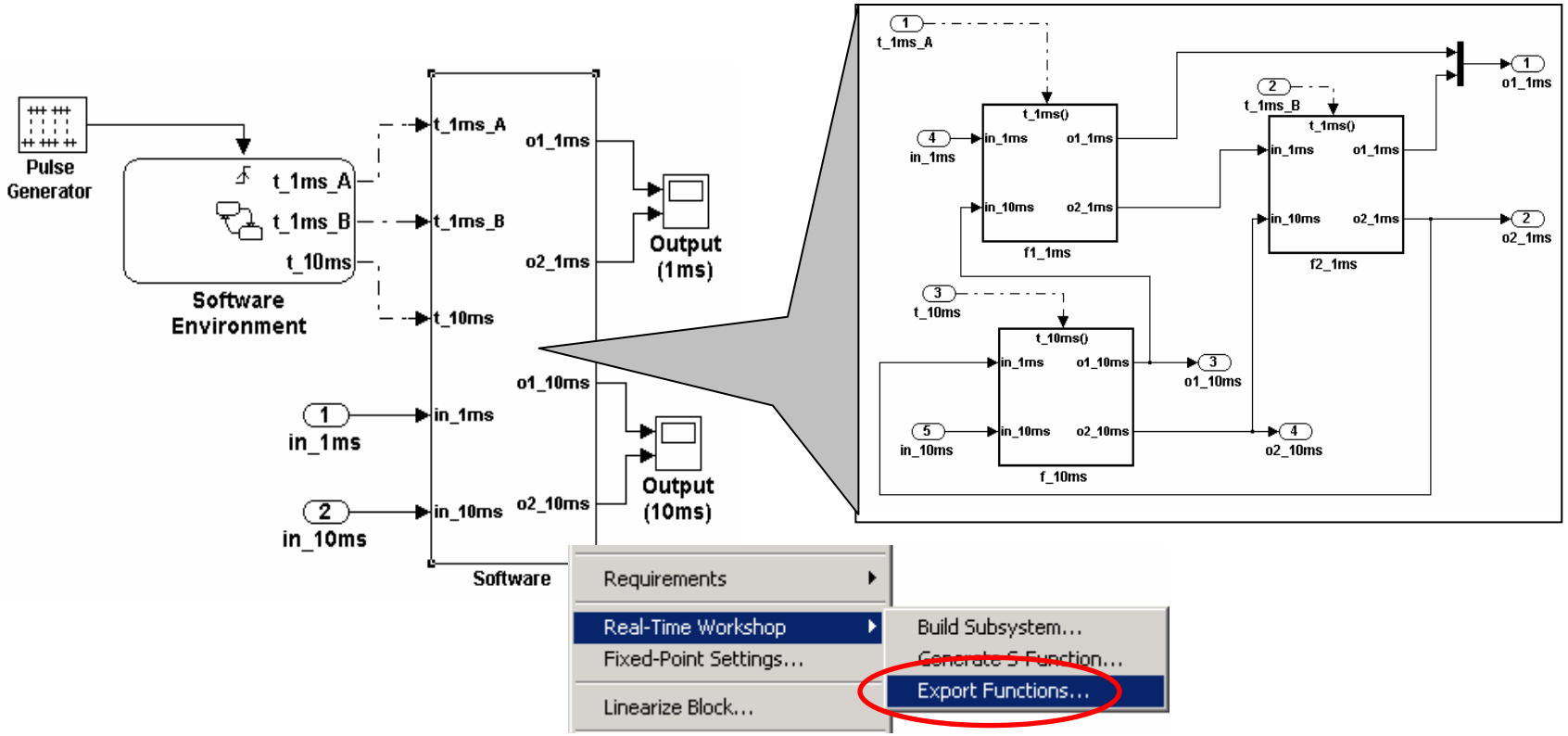
New features

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification

Code Architecture - Integration



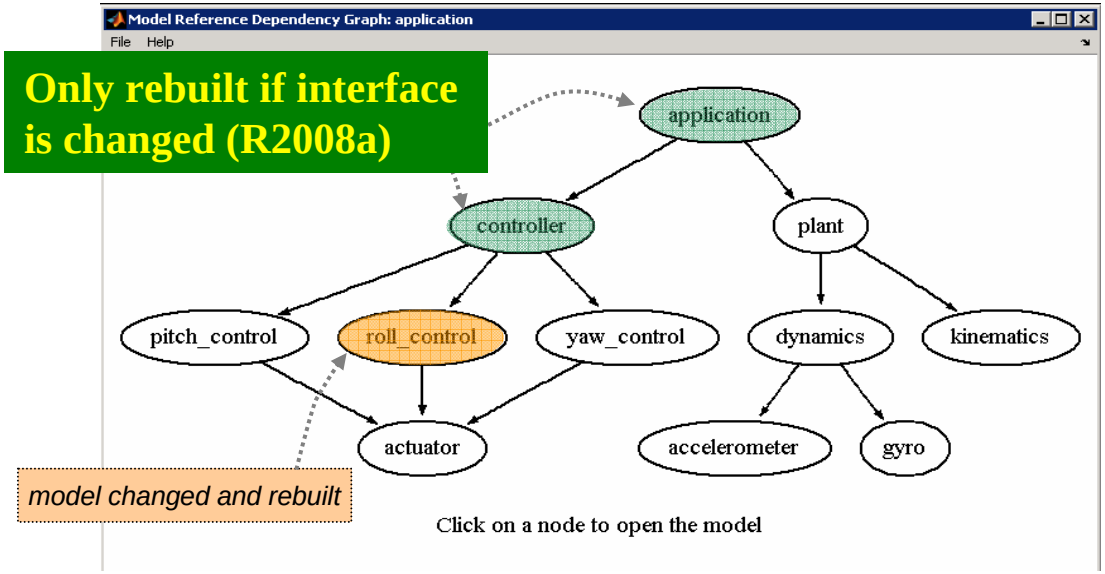
Subsystem Integration w/Export Functions – R2006a



- **Supports a popular scheduling technique in production**
- **Streamlines code generated**
  - **No scheduler, no model step function**

>> rtwdemo_export_functions

Model Reference – Introduced in Release 14



- Incremental code generation is supported via Model Reference
- When a model is changed, only the dependent model regenerate their code
- Normal mode model reference in R2007b

Model Integration w/Prototype Control – R2008a

Problem

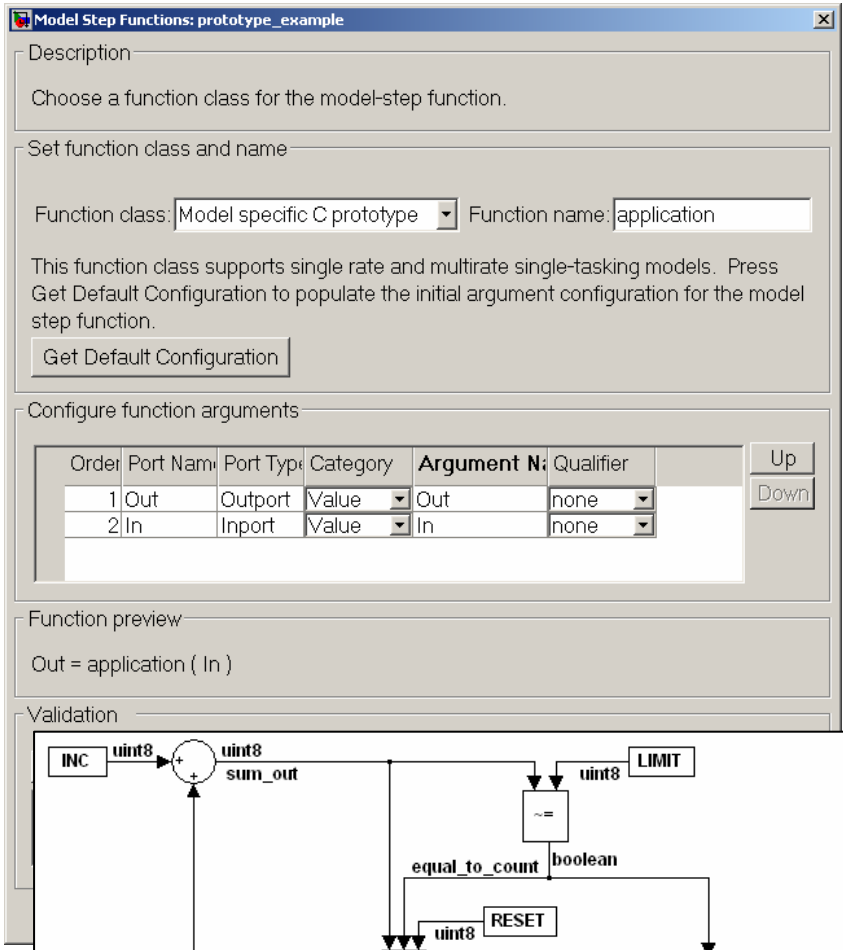
- Previously, non-reentrant code only supported void/void function interfaces

Solution

- Provide function control for **top model**
  - Pass inports/outports as arguments
  - Pass arguments by value or pointer
  - Control argument names and order
  - No wrapper
  - **Model Reference (R2008a)**

Benefit

- Simplifies code integration and testing
- Reduces global RAM usage

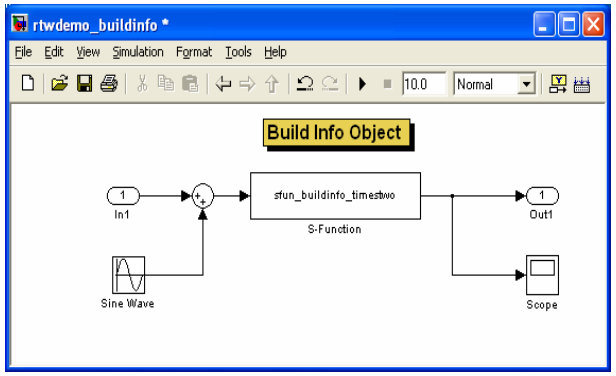



```
extern int32_T application(int32_T In);
```

Generated step function definition

Pack-and-Go – Introduced in R2006b

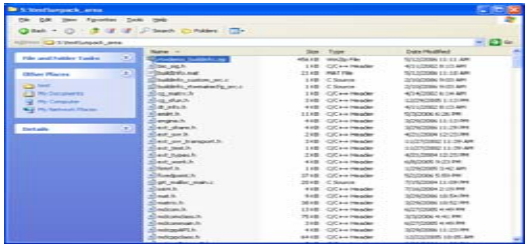
- Packages generated code and all dependencies (via zip)
- Includes Referenced Models – R2008a



Code, Pack

(Computer A)

Go



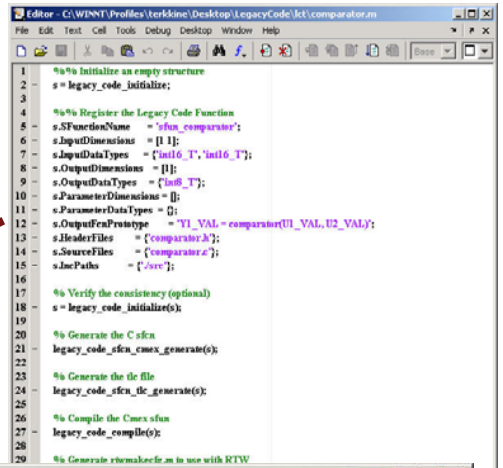
Unpack, Build

(Computer B)

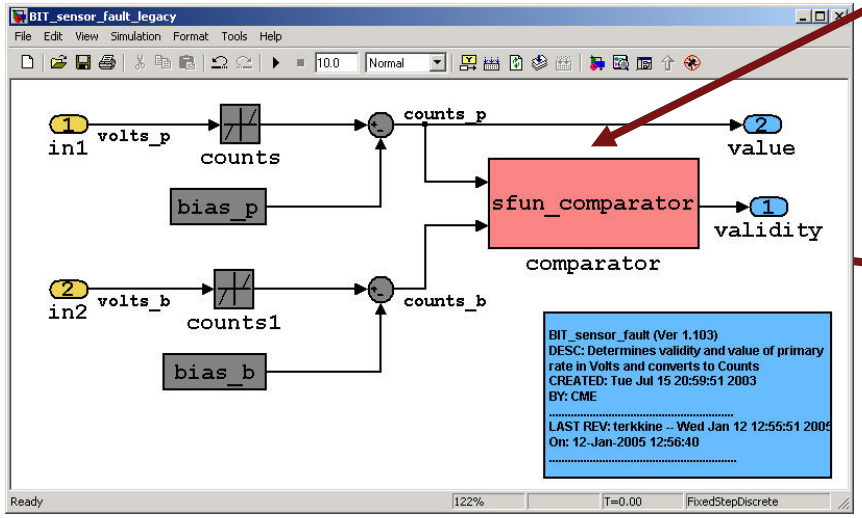
Legacy Code Tool – Introduced in R2006b

- Integrates external code for simulation and code generation (e.g., legacy lookup tables)
- Earlier version of LCT on MATLAB Central

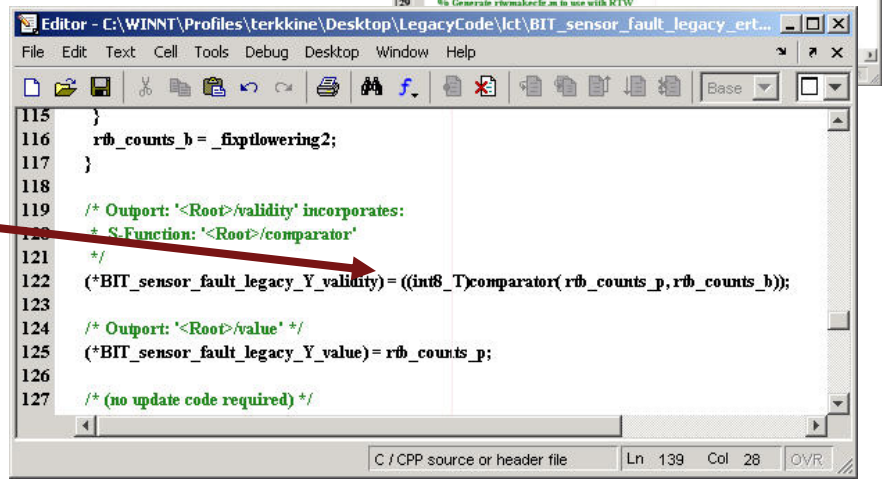
Registration File



Simulation Model



Generated Code



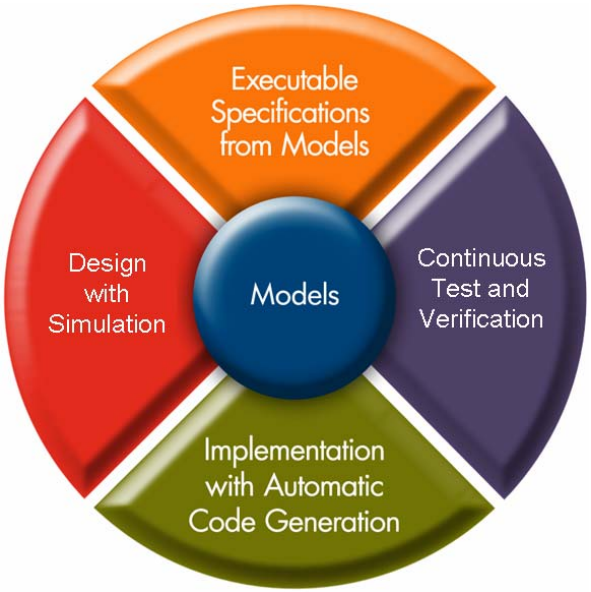

```
>> sldemo_lct_lut
```

Legacy Code Tool – R2007b

Enhanced for non-computational data integration (device driver, file descriptor,…)

	S-Function IO	S-Function Parameter	S-Function DWork
Built-in data types	Y	Y	Y
Simulink.Bus [1] (scalar only)	Y	N/A	Y
Simulink.AliasType [1]	Y	Y	Y
Simulink.NumericType [2]	Y	Y	Y
Fixed-point data types [3]	Y	Y	Y
Fi object	N/A	Y	N/A
Complex number [4]	Y	Y	Y
Scalar, 1D array, and 2D array	Y	Y	Y
ND array	Y	Y	Y
Void* & Void **	N	N	**R2007b**

Agenda



PCG Historical Review

- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features

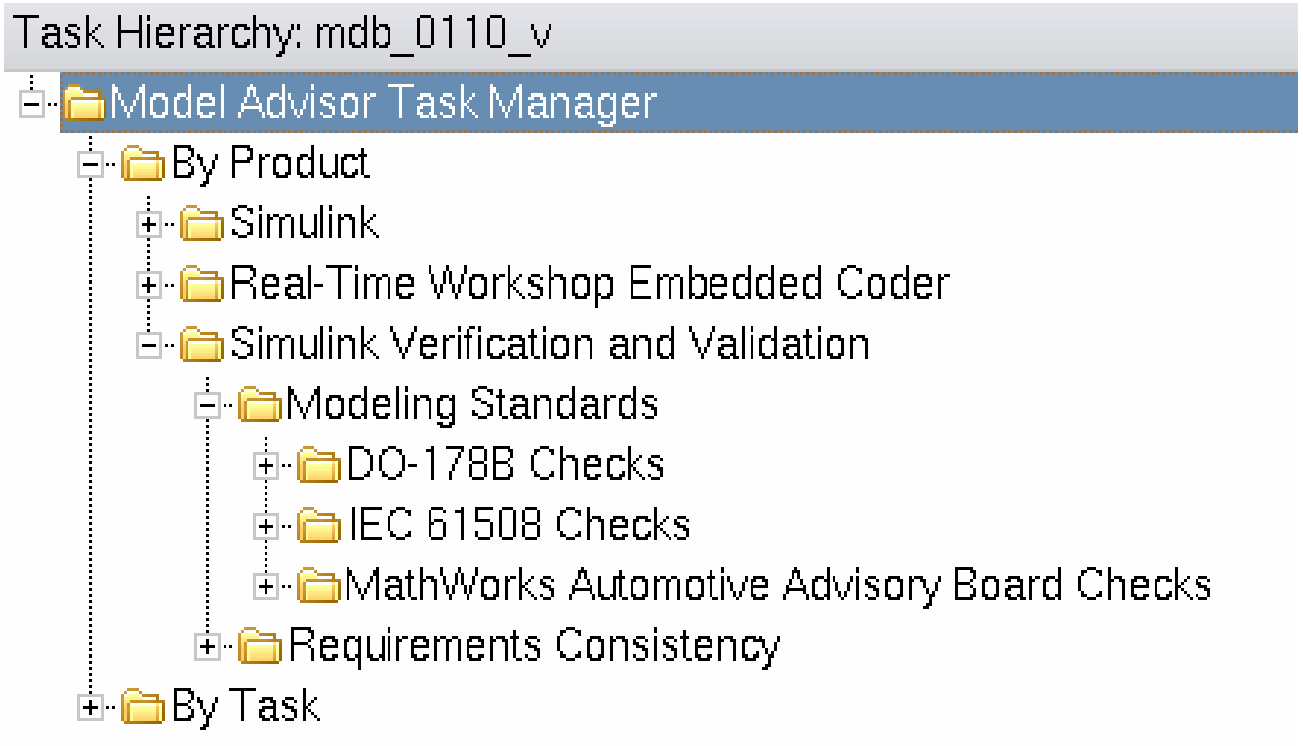
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Certification and Standards

Standards and Certification

- AUTOSAR (previously discussed)
- IEC 61508 Safety Guidelines (discussed tomorrow)
- MAAB Model Guidelines (v2)
- MISRA C® Code Guidelines (2004)

Model Standards Checks in Simulink Verification and Validation™

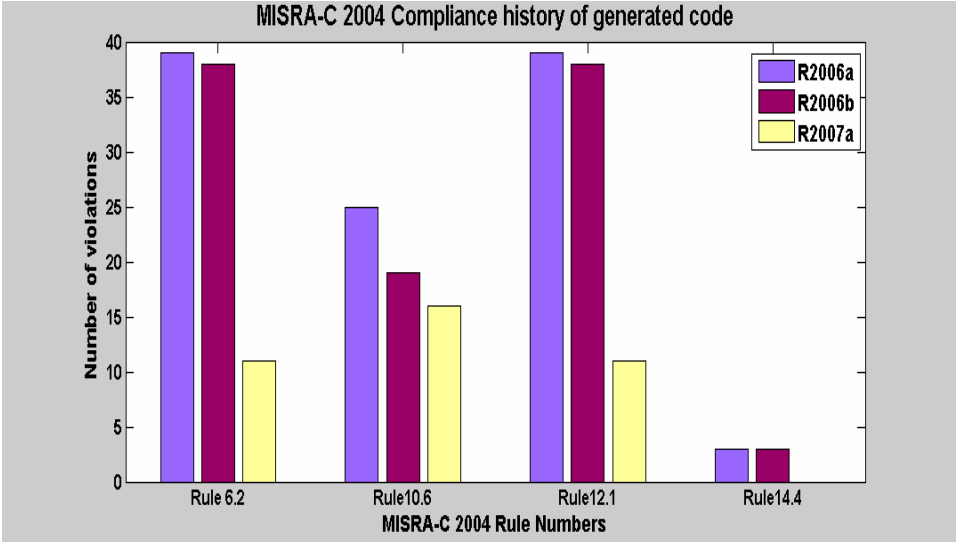
- MAAB v2 checks in R2007b
- IEC 61508 checks in R2008a



MISRA-C Compliance History of Generated Code

• **Our MISRA-C test suite consists of several example models**

• **Results shown for most frequently violated rules**



- Improving MISRA-C compliance with each release, such as:
  - Compliant code generated from enabled subsystems (R2007b)
  - Eliminate Stateflow® `goto` statements (R2007a)
  - Compliant parentheses option available (R2006b)
  - Generate *default* case for `switch-case` statements (R2006b)
- MathWorks MISRA-C Compliance Package available upon request
  www.mathworks.com/support/solutions/data/1-1IFP0W.html

Stateflow and Embedded MATLAB Traceability – R2008a

Problem

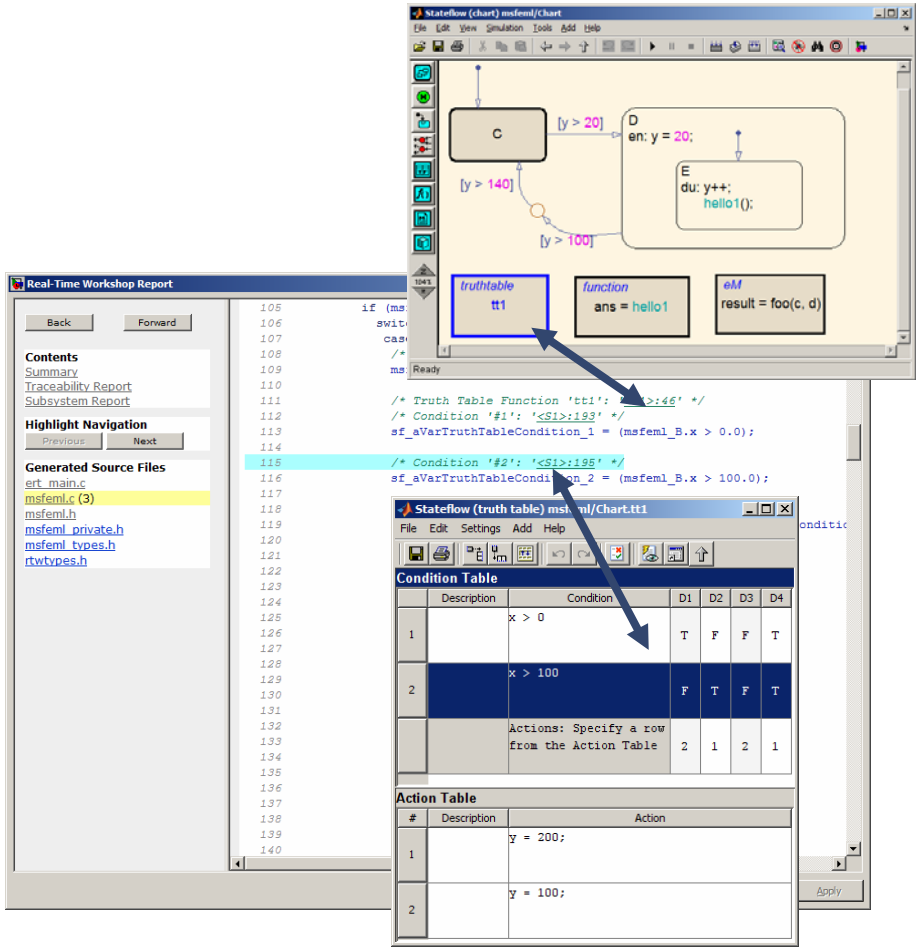
- No automated way to trace Stateflow chart and Embedded MATLAB code to generated code

Solution

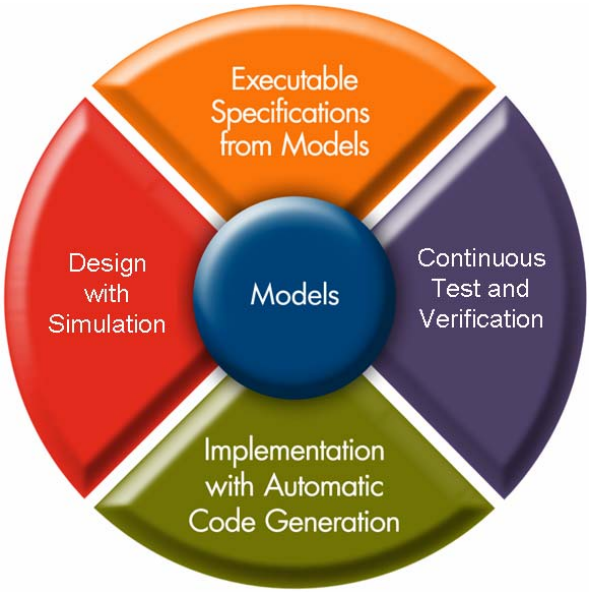
- Generate comments that automatically link Stateflow objects and Embedded MATLAB functions to the generated code
- Allow model-to-code and code-to-model bidirectional navigation

Benefit

- Bidirectional traceability helps code reviews, code verification, and software certification



Agenda



PCG Historical Review

- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

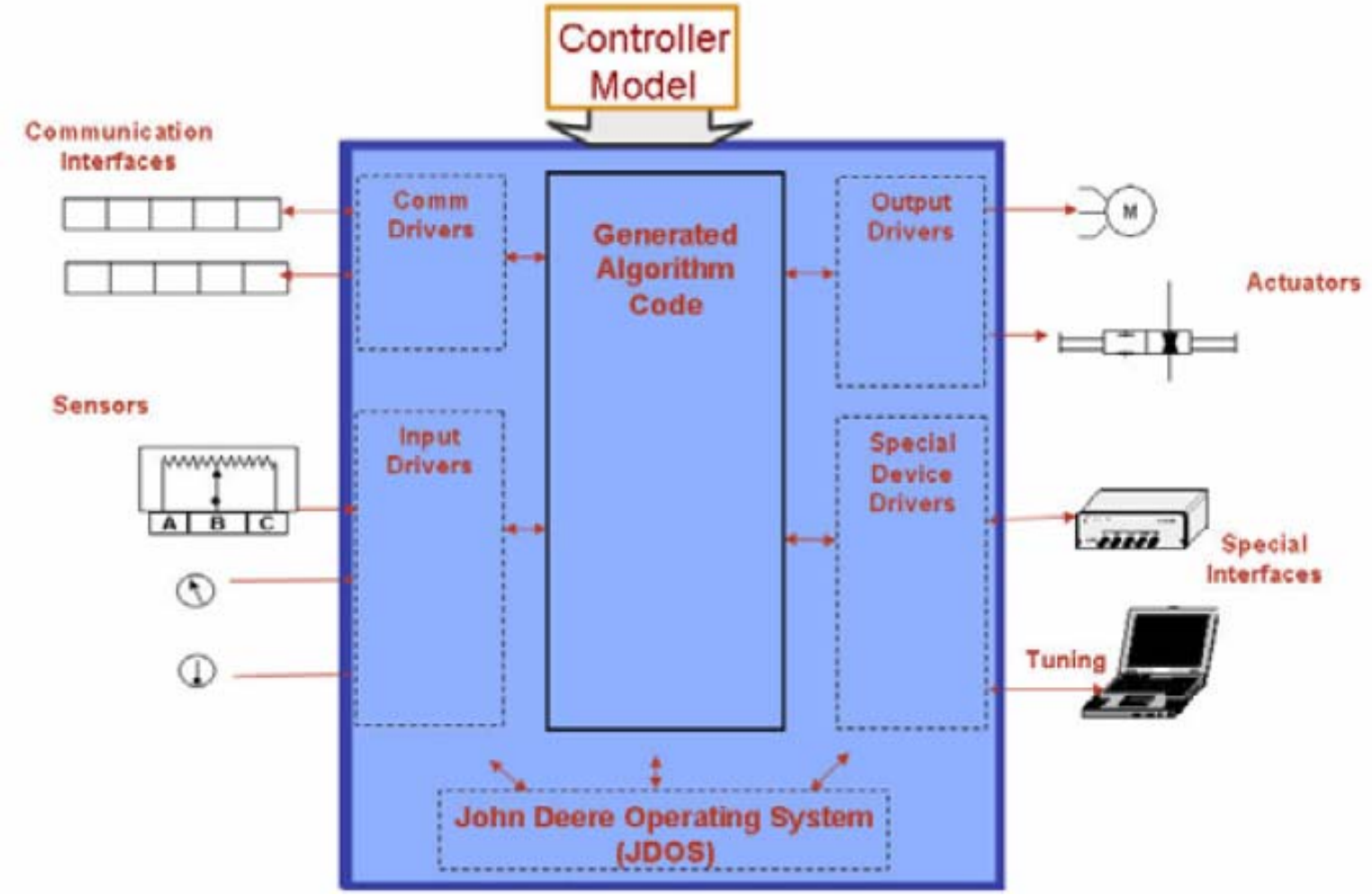
New Features

- Executable Specification
- Detailed Design
- Code Generation and Integration
- Certification and Standards

The Future?

Way Forward Machine

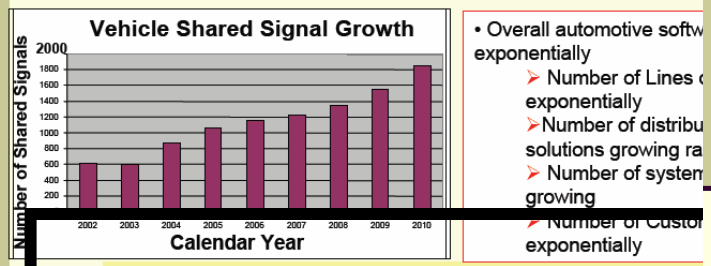
John Deere Production Code Generation - SAE CV 07



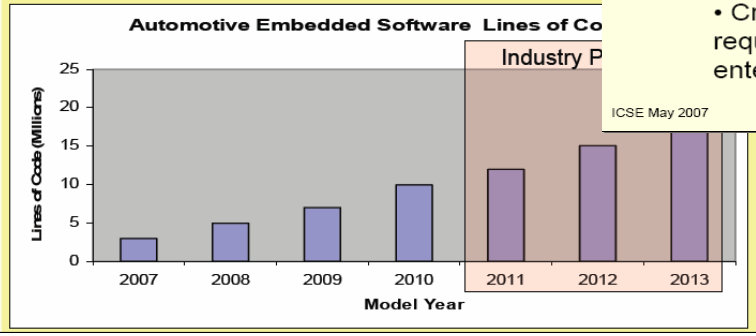
www.mathworks.com/mason/tag/proxy.html?dataid=9939&fileid=44540

Way Forward Machine

1.1 Technical Challenges - Softwar
Complexity is Growing Rapidl



• Overall automotive softw
exponentially
➢ Number of Lines
exponentially
➢ Number of distribu
solutions growing ra
➢ Number of systen
growing
➢ Number of Custo
exponentially



ICSE May 20

Pa 4

6.0 Conclusions

- Automotive software systems complexity is increasing exponentially
- Consumer electronics lifecycles will only accelerate this complexity growth
- Model Based design techniques have been proven to significantly improve the quality and robustness of software systems delivery
- Resource constraints & competitive efficiency goals drive the need for risk based prioritisation of scarce resources
- Risk based scaling of Process, Methods and Tools (PMT) enable consistent, traceable and dynamically modifiable PMT allocations.
- Cross domain, cross discipline functionality delivery will increasingly require full lifecycle management disciplines supported by fully integrated enterprise wide PLM systems.

ICSE May 2007

Page 15

20 MLOC
in 2013

Christopher Davey, Ford Motor Co, et al.

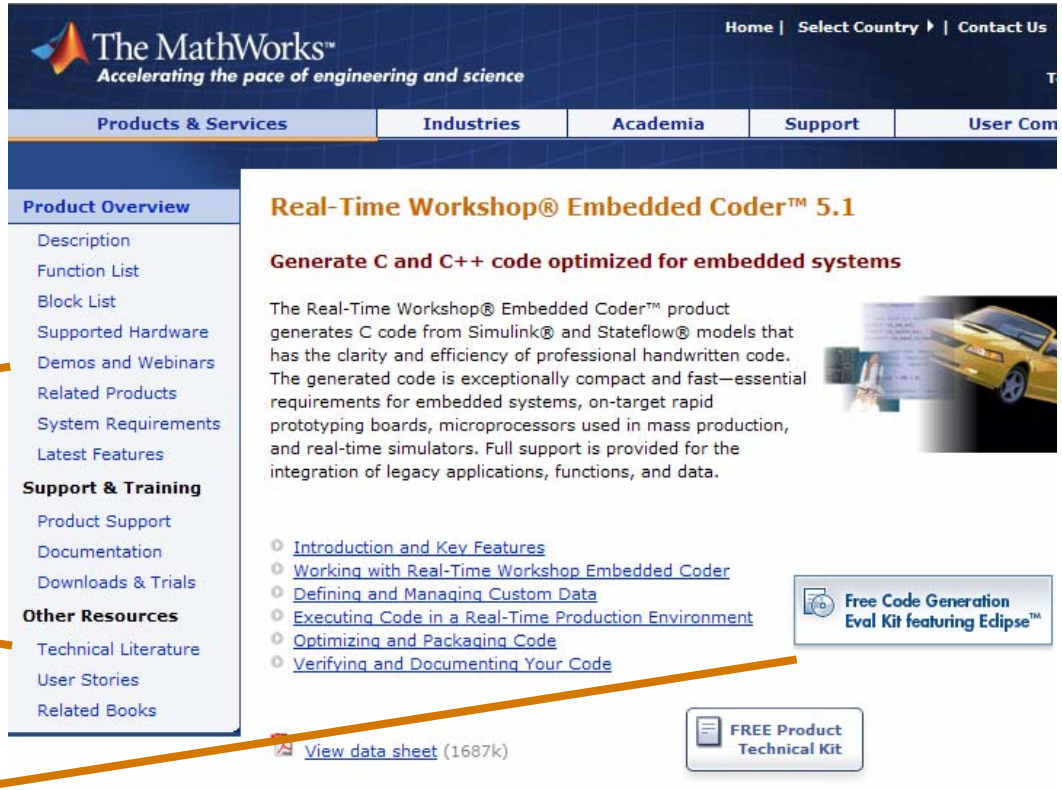
More Info – Product web page

Supported Hardware
- All links and targets
- Incl. Third Parties

Demos and Webinars
- Short web clips (2-5 minutes)

Technical Literature
- Conference papers

PCG Eval Kit for Eclipse (Europa)
- Step by step tuturial (3-4 hours)
- Earlier version in >>rtwdemos


The MathWorks™
Accelerating the pace of engineering and science
Home | Select Country | Contact Us
Products & Services
Industries
Academia
Support
User Com
Product Overview
Description
Function List
Block List
Supported Hardware
Demos and Webinars
Related Products
System Requirements
Latest Features
Support & Training
Product Support
Documentation
Downloads & Trials
Other Resources
Technical Literature
User Stories
Related Books
Real-Time Workshop® Embedded Coder™ 5.1
Generate C and C++ code optimized for embedded systems
The Real-Time Workshop® Embedded Coder™ product generates C code from Simulink® and Stateflow® models that has the clarity and efficiency of professional handwritten code. The generated code is exceptionally compact and fast—essential requirements for embedded systems, on-target rapid prototyping boards, microprocessors used in mass production, and real-time simulators. Full support is provided for the integration of legacy applications, functions, and data.
Introduction and Key Features
Working with Real-Time Workshop Embedded Coder
Defining and Managing Custom Data
Executing Code in a Real-Time Production Environment
Optimizing and Packaging Code
Verifying and Documenting Your Code
Free Code Generation Eval Kit featuring Eclipse™
View data sheet (1687k)
FREE Product Technical Kit

www.mathworks.com/rtwembedded

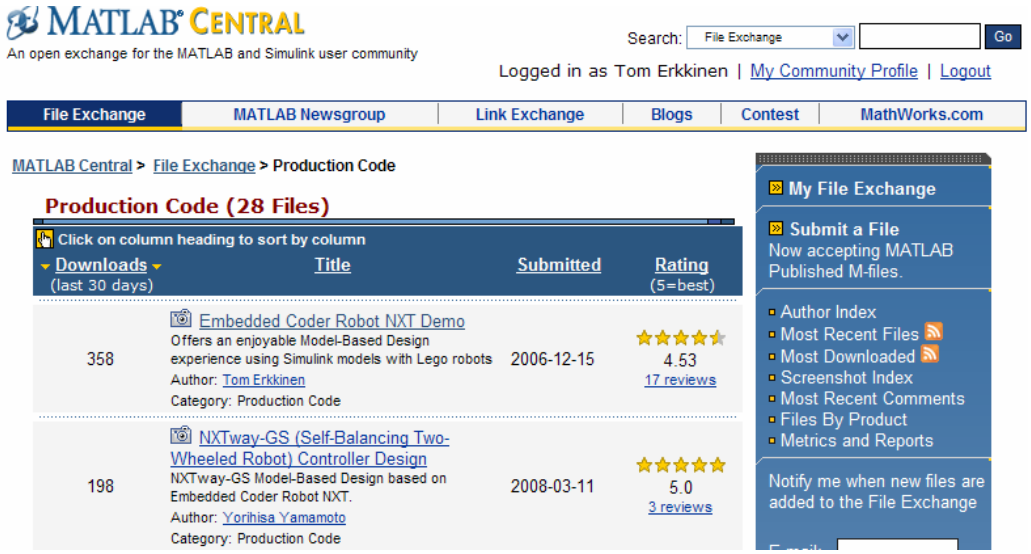
More Info – MATLAB Central

Demos

- Lego NXT Robots
- Task Profiling (5554)
- Dual Core (Blackfin)

Tools and Blocksets

- Code Coverage Tool
- Model Assistant Tool
- Target Blocksets



www.mathworks.com/matlabcentral/fileexchange

(Production Code category)

More Info – Support Examples

Categories

- Design Patterns
- Integration with Targets
- Scheduling and Multirate
- Standards and Guidelines

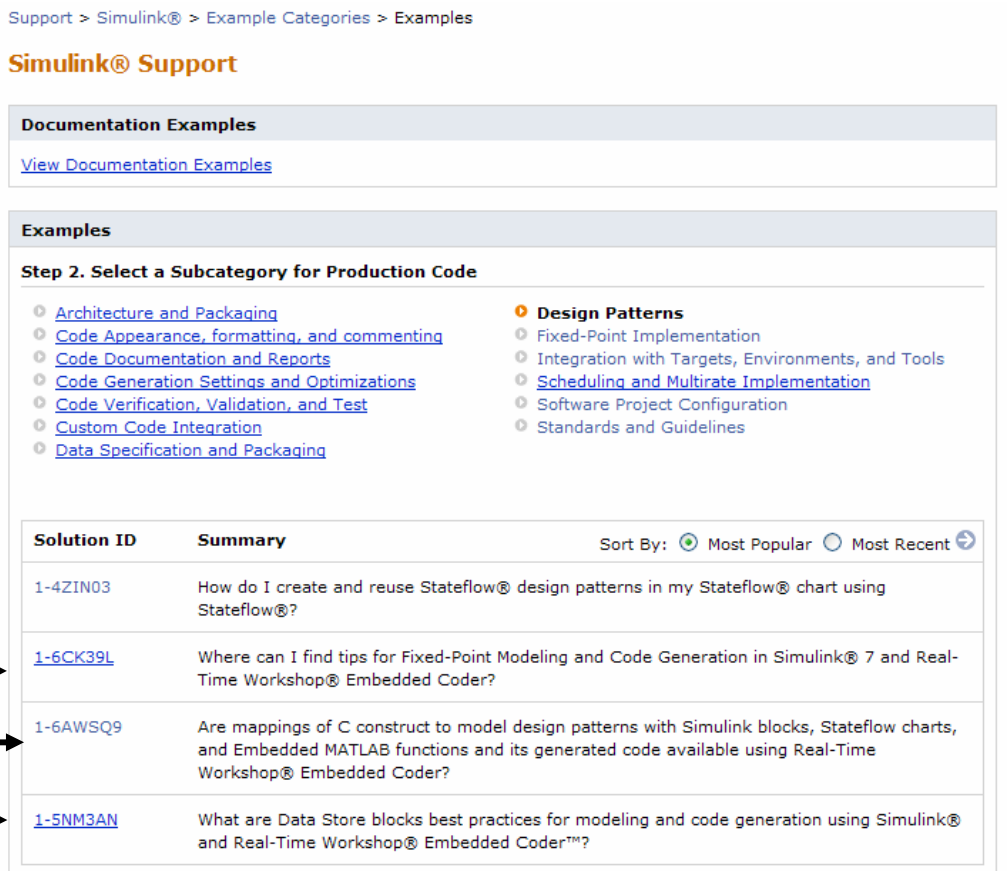
New for 2008a

Stateflow Pattern Wizard →

Fixed Point Tips for R2008a →

Model Patterns for C constructs →

Best Practices for Date Stores →



Support > Simulink® > Example Categories > Examples

Simulink® Support

Documentation Examples

View Documentation Examples

Examples

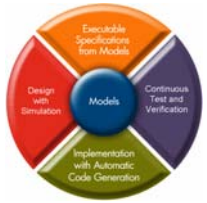
Step 2. Select a Subcategory for Production Code

- Architecture and Packaging
- Code Appearance, formatting, and commenting
- Code Documentation and Reports
- Code Generation Settings and Optimizations
- Code Verification, Validation, and Test
- Custom Code Integration
- Data Specification and Packaging
- **Design Patterns**
- Fixed-Point Implementation
- Integration with Targets, Environments, and Tools
- Scheduling and Multirate Implementation
- Software Project Configuration
- Standards and Guidelines

Solution ID	Summary	Sort By: Most Popular / Most Recent
1-4ZIN03	How do I create and reuse Stateflow® design patterns in my Stateflow® chart using Stateflow®?	
1-6CK39L	Where can I find tips for Fixed-Point Modeling and Code Generation in Simulink® 7 and Real-Time Workshop® Embedded Coder?	
1-6AWSQ9	Are mappings of C construct to model design patterns with Simulink blocks, Stateflow charts, and Embedded MATLAB functions and its generated code available using Real-Time Workshop® Embedded Coder?	
1-5NM3AN	What are Data Store blocks best practices for modeling and code generation using Simulink® and Real-Time Workshop® Embedded Coder™?	

www.mathworks.com/support/product/examples.html?product=SL&category=all

(Production Code category)



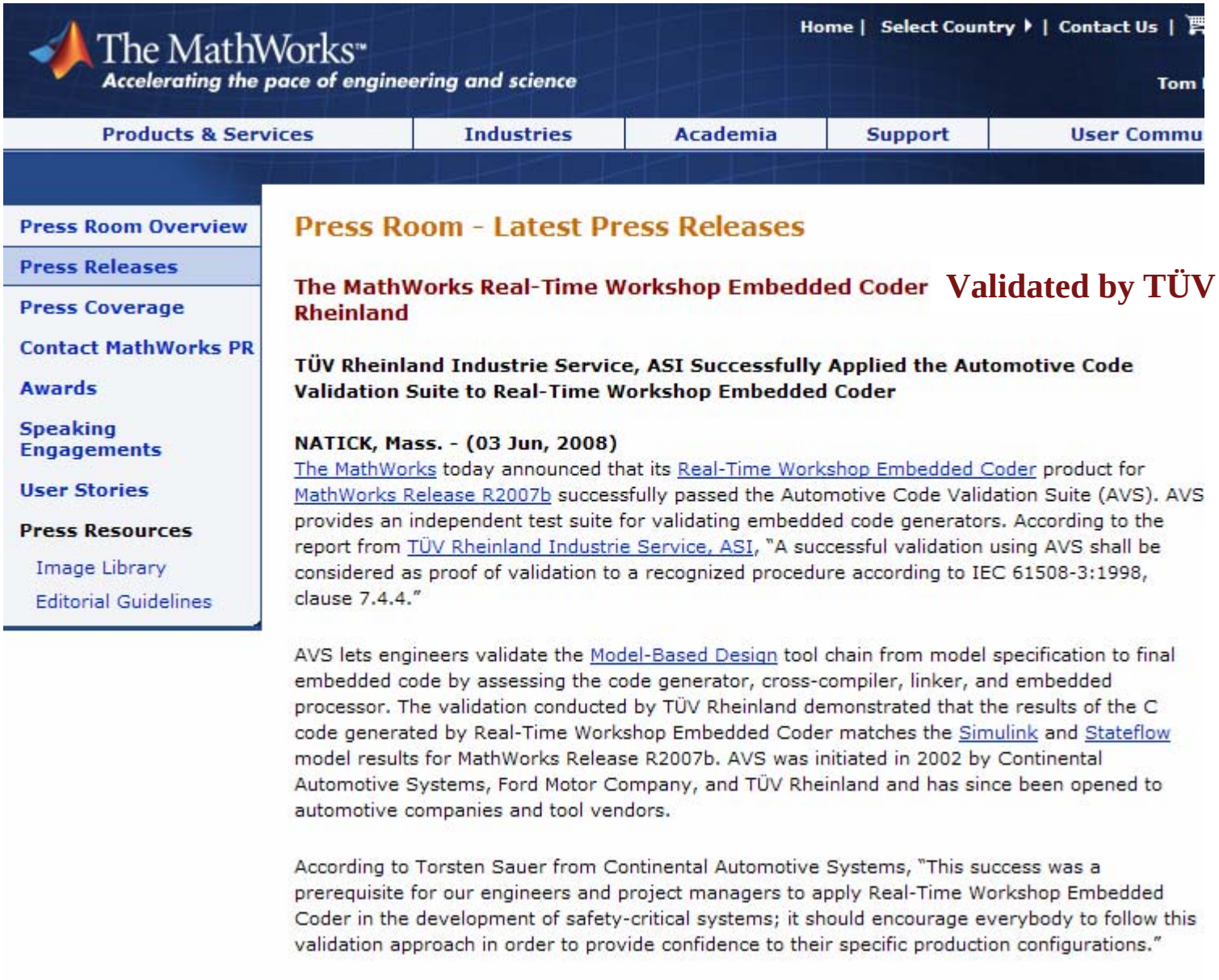
We are very interested in learning about your production code needs.

Thank you!

tom.erkkinen@mathworks.com

One More Thing …

Are you working on or planning for an IEC 61508 project?



The MathWorks™
Accelerating the pace of engineering and science

Home | Select Country | Contact Us |
Tom

Products & Services | Industries | Academia | Support | User Commu

- Press Room Overview
- Press Releases
- Press Coverage
- Contact MathWorks PR
- Awards
- Speaking Engagements
- User Stories
- Press Resources
  - Image Library
  - Editorial Guidelines

Press Room - Latest Press Releases

The MathWorks Real-Time Workshop Embedded Coder Validated by TÜV Rheinland

TÜV Rheinland Industrie Service, ASI Successfully Applied the Automotive Code Validation Suite to Real-Time Workshop Embedded Coder

NATICK, Mass. - (03 Jun, 2008)
The MathWorks today announced that its Real-Time Workshop Embedded Coder product for MathWorks Release R2007b successfully passed the Automotive Code Validation Suite (AVS). AVS provides an independent test suite for validating embedded code generators. According to the report from TÜV Rheinland Industrie Service, ASI, "A successful validation using AVS shall be considered as proof of validation to a recognized procedure according to IEC 61508-3:1998, clause 7.4.4."

AVS lets engineers validate the Model-Based Design tool chain from model specification to final embedded code by assessing the code generator, cross-compiler, linker, and embedded processor. The validation conducted by TÜV Rheinland demonstrated that the results of the C code generated by Real-Time Workshop Embedded Coder matches the Simulink and Stateflow model results for MathWorks Release R2007b. AVS was initiated in 2002 by Continental Automotive Systems, Ford Motor Company, and TÜV Rheinland and has since been opened to automotive companies and tool vendors.

According to Torsten Sauer from Continental Automotive Systems, "This success was a prerequisite for our engineers and project managers to apply Real-Time Workshop Embedded Coder in the development of safety-critical systems; it should encourage everybody to follow this validation approach in order to provide confidence to their specific production configurations."